

AD-A153 989

SPECIFICATION OF SOFTWARE QUALITY ATTRIBUTES VOLUME 2
SOFTWARE QUALITY SP. (U) BOOING AEROSPACE CO SEATTLE WA
T P BOWEN ET AL. FEB 85 D182-11678-2

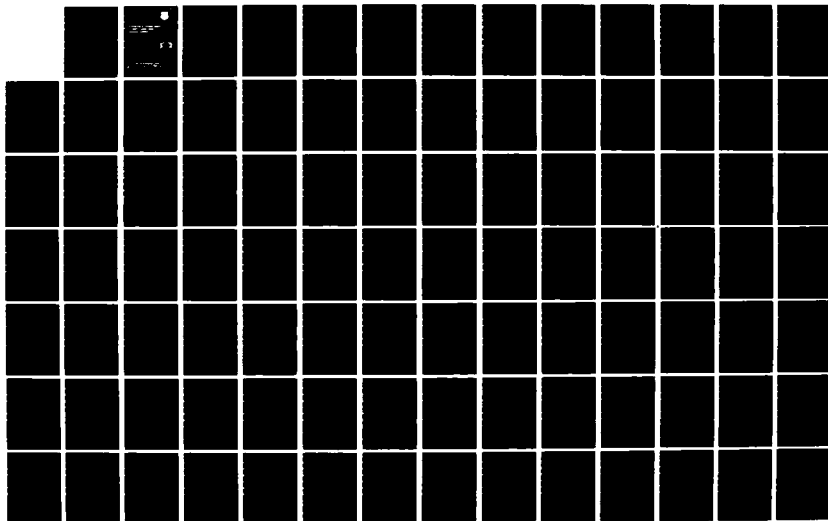
1/2

UNCLASSIFIED

RADC-TR-85-37-VOL-2 F30602-82-C-0137

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A153 989

Boeing

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-85-37, Volume II (of three) has been reviewed and is approved for publication.

APPROVED:



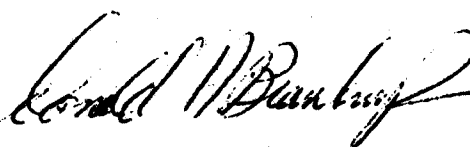
ROGER B. PANARA
Project Engineer

APPROVED:



RAYMOND P. URTZ, JR.
Technical Director
Command & Control Division

FOR THE COMMANDER:



DONALD A. BRANTINGHAM
Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COEE) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS N/A							
2a. SECURITY CLASSIFICATION AUTHORITY N/A		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.							
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A									
4. PERFORMING ORGANIZATION REPORT NUMBER(S) 182-11678-2		5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-85-37, Vol II (of three)							
6a. NAME OF PERFORMING ORGANIZATION Boeing Aerospace Company	6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COEE)							
6c. ADDRESS (City, State and ZIP Code) P.O. Box 3999 Seattle WA 98124		7b. ADDRESS (City, State and ZIP Code) Griffiss AFB NY 13441-5700							
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center	8b. OFFICE SYMBOL (If applicable) COEE	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-82-C-0137							
8c. ADDRESS (City, State and ZIP Code) Griffiss AFB NY 13441-5700		10. SOURCE OF FUNDING NOS. <table border="1"><tr><td>PROGRAM ELEMENT NO. 63728F</td><td>PROJECT NO. 2527</td><td>TASK NO. 03</td><td>WORK UNIT NO. 05</td></tr></table>		PROGRAM ELEMENT NO. 63728F	PROJECT NO. 2527	TASK NO. 03	WORK UNIT NO. 05		
PROGRAM ELEMENT NO. 63728F	PROJECT NO. 2527	TASK NO. 03	WORK UNIT NO. 05						
11. TITLE (Include Security Classification) SPECIFICATION OF SOFTWARE QUALITY ATTRIBUTES Software Quality Specification Guidebook									
12. PERSONAL AUTHOR(S) Thomas P. Bowen, Gary B. Wigle, Jay T. Tsai									
13a. TYPE OF REPORT Final	13b. TIME COVERED FROM Aug 82 to Oct 84	14. DATE OF REPORT (Yr., Mo., Day) February 1985	15. PAGE COUNT 156						
16. SUPPLEMENTARY NOTATION N/A									
17. COSATI CODES <table border="1"><tr><td>FIELD</td><td>GROUP</td><td>SUB. GR.</td></tr><tr><td>09</td><td>02</td><td></td></tr></table>		FIELD	GROUP	SUB. GR.	09	02		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Software Quality Software Quality Metrics	
FIELD	GROUP	SUB. GR.							
09	02								
19. ABSTRACT (Continue on reverse if necessary and identify by block number) Volume I (of three) describes the results and presents recommendations for integrating the RADC developed software quality metrics technology into the Air Force software acquisition management process and for changing Air Force acquisition documentation. In addition, changes to the baseline software quality framework are presented and features of a proposed specification methodology are summarized. Terminology and life cycle phases are consistent with the December 1983 draft of the DOD-STD-SDS, Defense System Software Development. Volume II (of three) describes how the software acquisition manager specifies software quality requirements, consistent with needs. Factor interrelationships, tradeoff among factor quality levels in terms of relative costs and an example for a command and control application are described. Procedures for assessing compliance with the specified requirements based on an analysis of data collected using procedures described in Volume III are included.									
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input checked="" type="checkbox"/> DTIC USERS <input type="checkbox"/>		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED							
22a. NAME OF RESPONSIBLE INDIVIDUAL Roger B. Panara		22b. TELEPHONE NUMBER (Include Area Code) (315) 330-4654	22c. OFFICE SYMBOL RADC (COEE)						

DD FORM 1473, 83 APR

EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

Volume III (of three) describes procedures and techniques for evaluating achieved quality levels. Worksheets for use in metric data collection by software life cycle phases and scoresheets for scoring each factor are provided.

DTIC
ELECTE
MAY 22 1985
B

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist.	Avail and/or Special
A-1	



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

PREFACE

This document is the second of three volumes of the Final Technical Report (CDRL ~~A004~~) for the Specification of Software Quality Attributes contract, F30602-82-C-0137. Contract work was performed by Boeing Aerospace Company (BAC) for Rome Air Development Center (RADC) to provide methods, techniques, and guidance to Air Force software acquisition managers ~~who specify the requirements for software quality.~~

The purpose of this contract was to (1) consolidate results of previous RADC contracts dealing with software quality measurement, (2) enhance the software quality framework, and (3) develop a methodology to enable a software acquisition manager to determine and specify software quality factor requirements. We developed the methodology and framework elements ^{was developed} to focus on an Air Force software acquisition manager specifying quality requirements for embedded software that is part of a command and control application. This methodology and most of the framework elements are generally useful for other applications and different environments.

The Final Technical Report consists of three volumes:

- a. Volume I, Specification of Software Quality Attributes—Final Report.
- b. Volume II, Specification of Software Quality Attributes—Software Quality Specification Guidebook.
- c. Volume III, Specification of Software Quality Attributes—Software Quality Evaluation Guidebook.

Volume I describes the results of research efforts conducted under this contract, including recommendations for integrating quality metrics technology into the Air Force software acquisition management process, recommended changes to Air Force software acquisition documentation, and summaries of software quality framework changes and specification methodology features.

Volumes II and III describe the methodology for using the quality metrics technology and include an overview of the software acquisition process using this technology and the quality framework. Volume II describes methods for specifying software quality requirements and addresses the needs of the software acquisition manager. Volume III

describes methods for evaluating achieved quality levels of software products and addresses the needs of data collection and analysis personnel.

Volume II also describes procedures and techniques for specifying software quality requirements in terms of quality factors and criteria. Factor interrelationships, relative costs to develop high quality levels, and an example for a command and control application are also described. Procedures for assessing compliance with specified requirements are included.

Volume III also describes procedures and techniques for evaluating achieved quality levels of software products. Worksheets for collecting metric data by software life-cycle phase and scoresheets for scoring each factor are provided in the appendixes. Detailed metric questions on worksheets are nearly identical to questions in the Software Evaluation Reports proposed as part of the Software Technology for Adaptable Reliable Systems (STARS) Measurement data item descriptions (DID).

Terminology and life-cycle phases used in the guidebooks are consistent with the December 1983 draft of the Department of Defense software development standard (DOD-STD-SDS) (e.g., the term computer software configuration item (CSCI) is used rather than computer program configuration item (CPCI)).

CONTENTS

	<u>Page</u>
1.0 INTRODUCTION	1-1
1.1 Background	1-1
1.2 Purpose	1-2
1.3 Scope	1-2
1.4 Use of the Guidebooks	1-3
2.0 ROLE OF QUALITY METRICS IN THE SOFTWARE ACQUISITION PROCESS	2-1
2.1 Software Acquisition Process	2-1
2.1.1 System Acquisition Life Cycle	2-1
2.1.2 Software Development Cycle	2-3
2.1.3 Life-Cycle Relationships	2-5
2.1.4 Software Acquisition Management	2-7
2.1.5 Verification and Validation	2-8
2.1.6 Quality Assurance	2-9
2.2 Quality Metrics	2-11
2.2.1 Framework	2-15
2.2.2 Quality Specification	2-21
2.2.3 Quality Monitoring	2-23
2.3 Software Acquisition Using Quality Metrics	2-25
2.4 Potential Benefits and Problems	2-33
2.4.1 Benefits	2-33
2.4.2 Problems	2-34
3.0 QUALITY METRICS FRAMEWORK	3-1
3.1 Software Quality Factors	3-3
3.1.1 Factor Definitions and Rating Formulas	3-3
3.1.2 Quality Factor Interrelationships	3-9
3.2 Software Quality Criteria	3-11
3.3 Software Quality Metrics	3-11

	<u>Page</u>
3.4 Metric Worksheets	3-15
3.5 Factor Scoresheets	3-17
 4.0 SOFTWARE QUALITY SPECIFICATION METHODOLOGY	 4-1
4.1 Select and Specify Quality Factors	4-7
4.1.1 Identify Functions (Step 1)	4-9
4.1.2 Assign Quality Factors and Goals (Step 2)	4-11
4.1.2.1 Command and Control Quality Concerns	4-11
4.1.2.2 System Quality Factors	4-11
4.1.2.3 Quality Requirements Survey	4-19
4.1.2.4 Complementary Quality Factors	4-19
4.1.2.5 Quality Goals Assignment	4-21
4.1.3 Consider Interrelationships (Step 3)	4-23
4.1.3.1 Shared Criteria	4-25
4.1.3.2 Beneficial and Adverse Relationships	4-25
4.1.3.3 Quantification of Relationships	4-27
4.1.3.4 Review of Quality Goals	4-35
4.1.4 Consider Costs (Step 4)	4-39
4.1.4.1 Life-Cycle Quality Costs and Benefits	4-39
4.1.4.2 Cost Variation Estimates	4-49
4.1.4.3 Cost Effects of Factor Interrelationships	4-51
4.1.4.4 Review of Quality Goals	4-61
4.2 Select and Specify Quality Criteria	4-63
4.2.1 Select Criteria (Step 1)	4-63
4.2.2 Assign Weighting Formulas (Step 2)	4-63
4.2.3 Consider Interrelationships (Step 3)	4-65
4.3 Select and Qualify Quality Metrics	4-67
4.3.1 Identify Metrics (Step 1)	4-67
4.3.2 Select and Qualify Metric Elements (Step 2)	4-68
4.4 Assess Compliance with Requirements	4-69
4.4.1 Review Requirements Allocations and Evaluation Formulas	4-69
4.4.2 Review Factor Scores	4-70
4.4.3 Review Criteria Scores	4-71
4.4.4 Review Metric Scores	4-72

	<u>Page</u>
Appendix A—Metric Worksheets	A-1
Appendix B—Factor Scoresheets	B-1
Appendix C—Software Quality Evaluation Report	C-1

FIGURES

	<u>Page</u>
1.4-1 Software Quality Measurement Methodology	1-4
2.1-1 System Acquisition Life-Cycle Phases and Decision Points	2-2
2.1-2 Software Development Cycle	2-4
2.1-3 Life-Cycle Relationship between the System and the Operational Software	2-6
2.1-4 Relationship of Software Development and V&V	2-10
2.1-5 Software QA Function	2-12
2.2-1 Quality Metrics Technology—Life-Cycle Model	2-14
2.2-2 Software Quality Model	2-16
2.2-3 Performance Factor Attributes	2-18
2.2-4 Design Factor Attributes	2-19
2.2-5 Adaptation Factor Attributes	2-20
2.3-1 Software Acquisition Quality Metrics Functions	2-24
2.3-2 Air Force Acquisition Relationships Involved in Quality Metrics Functions	2-26
2.3-3 Recommended Responsibilities and Relationships for the QM Specification Function	2-28
2.3-4 Recommended Responsibilities and Relationships for the QM Monitoring Function	2-30
2.3-5 Relationship between Product Divisions and DACS	2-32
3.1-1 Rating Estimation and Rating Assessment Windows	3-4
4.0-1 Software Quality Specification and Evaluation Process	4-2
4.0-2 Flow of Software Quality Requirements	4-4
4.0-3 Procedures for Specifying Software Quality Requirements	4-6
4.0-4 Procedures for Assessing Compliance with Requirements	4-8
4.1.4-1 Quality Factor Life-Cycle Cost Ranges	4-40
4.1.4-2 Cost Effects of Positive Factor Interrelationships	4-52
4.1.4-3 Cost Effects of Negative Factor Interrelationships	4-56

TABLES

	<u>Page</u>
2.2-1 Quality Concerns	2-17
2.2-2 Software Quality Factor Interrelationships	2-22
2.3-1 Organizational Evaluation	2-29
3.1-1 Software Quality Factor Definitions and Rating Formulas	3-2
3.1-2 Quality Factor Ratings	3-6
3.2-1 Software Quality Factors and Criteria	3-10
3.2-2 Quality Criteria Definitions	3-12
3.3-1 Quality Metrics Summary	3-13
3.4-1 Metric Worksheet/Life-Cycle Correlation	3-16
3.4-2 Software Development Products	3-18
4.1.1-1 Characteristics and Functions for Example System	4-10
4.1.2-1 Important S/W Quality Factors for Major C ² Applications	4-12
4.1.2-2 Examples of Application/Environment Characteristics and Related Software Quality Factors	4-13
4.1.2-3 System/Software Quality Factor Correlation	4-14
4.1.2-4 Software Quality Requirements Survey	4-18
4.1.2-5 Software Quality Factor Identification Form—Survey Results	4-20
4.1.2-6 Complementary Software Quality Factors	4-22
4.1.2-7 Software Quality Factor Identification Form—Initial Goals	4-24
4.1.3-1 Effects of Criteria on Software Quality Factors	4-26
4.1.3-2 Positive Factor Interrelationships	4-28
4.1.3-3 Negative Factor Interrelationships	4-32
4.1.3-4 Factor Interrelationship Calculations	4-34
4.1.3-5 Software Quality Factor Identification Form—Revised Goals	4-36
4.1.4-1 Life Cycle Quality Costs/Benefits	4-38
4.1.4-2 Cost Variations Calculation Form—Initial Estimate	4-48
4.1.4-3 Cost Variations Calculation Form—Refined Estimate	4-60
4.1.4-4 Software Quality Factor Identification Form—Final Goals	4-62
4.2-1 Criteria Weighting Formula Form—Initial Weighting	4-64
4.2-2 Criteria Weighting Formula Form—Revised Weighting	4-66

GLOSSARY

AFCMD	Air Force Contracts Management Division
AFCL	Air Force Logistics Command
AFPRO	Air Force Plant Representative Office
APSE	Ada programming support environment
ASD	Aeronautical Systems Division
CDR	critical design review
CPCI	computer program configuration item
CSC	computer software component
CSCI	computer software configuration item
DACS	Data and Analysis Center for Software
DAE	Defense Acquisition Executive
DID	data item description
DOD	Department of Defense
DOD-STD-SDS	Department of Defense software development standard
DOD-STD-SQS	Department of Defense software quality standard
ESD	Electronic Systems Division
FCA	functional configuration audit
FSD	full-scale development
HOL	high order language
I/O	input/output
IV&V	independent validation and verification
PCA	physical configuration audit
PDR	preliminary design review
QA	quality assurance
QM	quality metrics
RADC	Rome Air Development Center
SD	Space Division
SDR	system design review
SPO	System Program Office
SSR	software specification review
STARS	Software Technology for Adaptable Reliable Systems
TRR	test readiness review
V&V	verification and validation

1.0 INTRODUCTION

1.1 BACKGROUND

There has been a recent, increased awareness of critical problems encountered in developing large-scale systems involving software. These problems include cost and schedule overruns, high cost sensitivity to changes in requirements, poor performance of delivered systems, high system-maintenance costs, and lack of reusability.

The government (the Department of Defense (DOD) in particular) as a customer for large-scale system developments, has sponsored efforts to address these problems; for example, development of Ada programming language and Ada programming support environments (APSE), proposed DOD standards for software development (DOD-STD-SDS) and quality (DOD-STD-SQS), the Software Technology for Adaptable Reliable Systems (STARS) program, proposed STARS measurement data item descriptions (DID), and various development aids and tools. These all provide partial solutions.

Since 1976, Rome Air Development Center (RADC) has pursued a program intended to achieve better control of software quality. Through a series of related contracts, this program has sought to identify key software quality issues and to provide a valid methodology for specifying software quality requirements and measuring achieved quality levels of software products released incrementally during the software life cycle. A quality model was established in which a hierarchical relationship exists between a user-oriented quality factor at the top level and software-oriented attributes at the second and third levels (criteria and metrics). Software quality is predicted and measured by the presence, absence, or degree of identifiable software attributes. (See Sec. 2.2 for an explanation of the quality model and an overview of quality factors and attributes.)

The Final Technical Report for this contract (F30602-82-C-0137) contains the most recent results of the RADC software quality program. This report incorporates pertinent results from and uses foundations established in previous contracts. The Final Technical Report consists of three volumes: the Final Report, the Software Quality Specification Guidebook, and the Software Quality Evaluation Guidebook.

1.2 PURPOSE

The purpose of this guidebook (Vol. II, Software Quality Specification Guidebook) is to provide a comprehensive set of procedures and techniques to enable an Air Force software acquisition manager to specify quality requirements for software embedded in command and control systems. Volume III, Software Quality Evaluation Guidebook, provides a comprehensive set of procedures and techniques to enable data collection personnel to apply quality metrics to software products and to evaluate the achieved quality levels. Volume I, Final Report, summarizes the results of contract task efforts.

The purpose of the quality metrics technology is to provide a more disciplined engineering approach to specifying, predicting, and evaluating software quality. The benefits of this approach include software life-cycle cost savings (or cost avoidance) and software products that reflect user-customer quality needs. Rigorous application of metrics at incremental releases of software products throughout the life cycle provides for early detection of quality-related problems. Periodic assessment of quality levels provides better management visibility and enables timely decision making.

1.3 SCOPE

Section 2.0 describes the role of quality metrics in the software acquisition process. Descriptions of the system acquisition life cycle and software development cycle are provided with a discussion of their relationships. Specifying quality requirements and monitoring software product quality levels are described within the life-cycle perspective. The software quality model and framework elements are introduced.

Section 3.0 describes quality framework terminology and concepts key to understanding subsequent details. All framework elements—factors, criteria, metrics, worksheets, and scoresheets—are also described.

Section 4.0 describes procedural steps for selecting and specifying quality requirements in terms of quality factors and criteria and for selecting metrics. Trade studies are identified to aid decision making, and, for clarification, an example for a

command and control application is continued throughout the procedural steps. The procedural steps for assessing compliance with specified software quality requirements also are described.

This guidebook incorporates pertinent results from previous research concerning software quality measurement conducted for RADC. Results of this research are described in Software Quality Measurement for Distributed Systems, RADC-TR-83-175, Volumes I, II, and III. Software life-cycle phases and terminology used throughout this guidebook are consistent with the December 1983 draft of DOD-STD-SDS. Significant enhancements to previous contract results are noted in the following paragraphs.

The software quality model (described in Sec. 2.2) addresses software quality at three hierarchical levels: quality factors, criteria, and metrics. The methodology described in Section 4.0 uses same three categories. The hierarchical levels of quality parallel the chronological procedural steps in the methodology (i.e., factors, criteria, and metrics). Procedural steps are detailed and include consideration of interrelationships among quality factors and relative costs to develop high quality levels.

Framework elements are also enhanced. Factors are categorized under performance, design, and adaptation to aptly indicate acquisition concerns. Criteria are organized under the same three acquisition concerns, thereby simplifying the attribute relationships. Metric questions on the worksheets include explanatory information and formulas and are nearly identical to the questions in the Software Evaluation Reports proposed as part of the STARS measurement DIDs.

1.4 USE OF THE GUIDEBOOKS

This Software Quality Specification Guidebook addresses the needs of Air Force software acquisition managers. Procedures are provided for specifying quality requirements and for assessing compliance with requirements. The Software Quality Evaluation Guidebook (see Vol. III) addresses the needs of personnel collecting and analyzing metric data. Procedures are provided for applying metrics, generating metric scores, analyzing scoring, and reporting results.

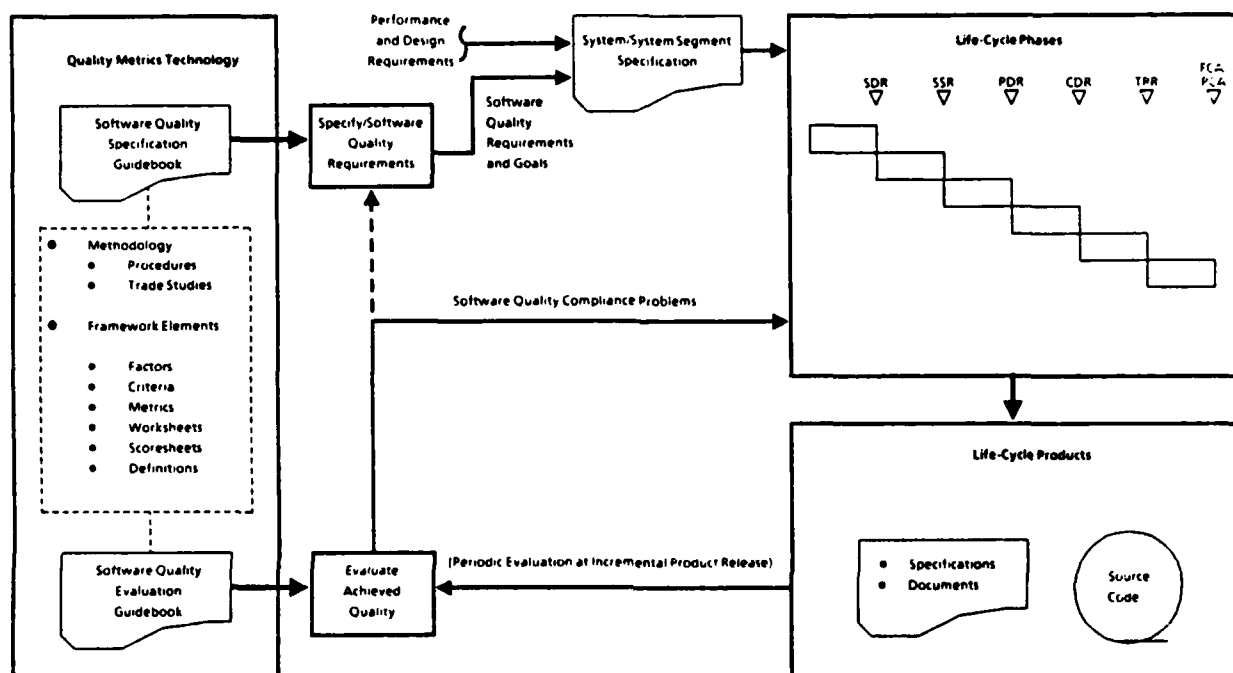


Figure 1.4-1 Software Quality Measurement Methodology

Procedures in each guidebook are contained in Section 4.0. Sections 1.0, 2.0, and 3.0 contain nearly identical information on the elements, perspective, and role of quality metrics technology.

The guidebooks were designed for use with new projects, in which procedures are performed (primarily) chronologically throughout system and software life cycles as depicted in Figure 1.4-1. Using quality metrics technology and guidebooks for evaluating system and software products in other contexts is addressed in Section 4.0. Detailed explanations of life-cycle phases, review points, framework elements, and methodology are provided in Sections 2.0, 3.0, and 4.0.

2.0 ROLE OF QUALITY METRICS IN THE SOFTWARE ACQUISITION PROCESS

This section examines elements of Air Force system acquisition and software acquisition processes, describes the process used for specifying and monitoring quality levels, and discusses the role of quality metrics (QM) technology in the Air Force software acquisition management process. Considerations include how QM technology can be integrated into the Air Force software acquisition process and how existing mechanisms within the acquisition process can be used to implement QM technology. Advantages and disadvantages of using QM technology in software acquisition management and of integrating QM technology into the software acquisition management process are also discussed.

2.1 SOFTWARE ACQUISITION PROCESS

The following sections describe selected concepts associated with Air Force software acquisition management, including system acquisition life cycle, software development cycle, life-cycle relationships, software acquisition management, verification and validation (V&V), and quality assurance (QA). Concepts introduced here provide a basis for discussions of QM technology integration and implementation in the acquisition process in later Sections. The system acquisition life cycle and software development cycle are fully defined in DODD 5000.1 and DOD-STD-SDS and are only summarized here. This Section is not intended to describe all activities of each life-cycle phase but to establish the background for discussion of the role of QM technology.

2.1.1 System Acquisition Life Cycle

The system acquisition life cycle defined in DOD-STD-SDS consists of four phases: concept exploration, demonstration and validation, full-scale development (FSD), and production and deployment. Four major decision points are associated with these phases as shown in Figure 2.1-1 and as defined in DODD 5000.1 (Major System Acquisition). These points are mission need determination; concept selection, milestone I; program go-ahead, milestone II; and production and deployment, milestone

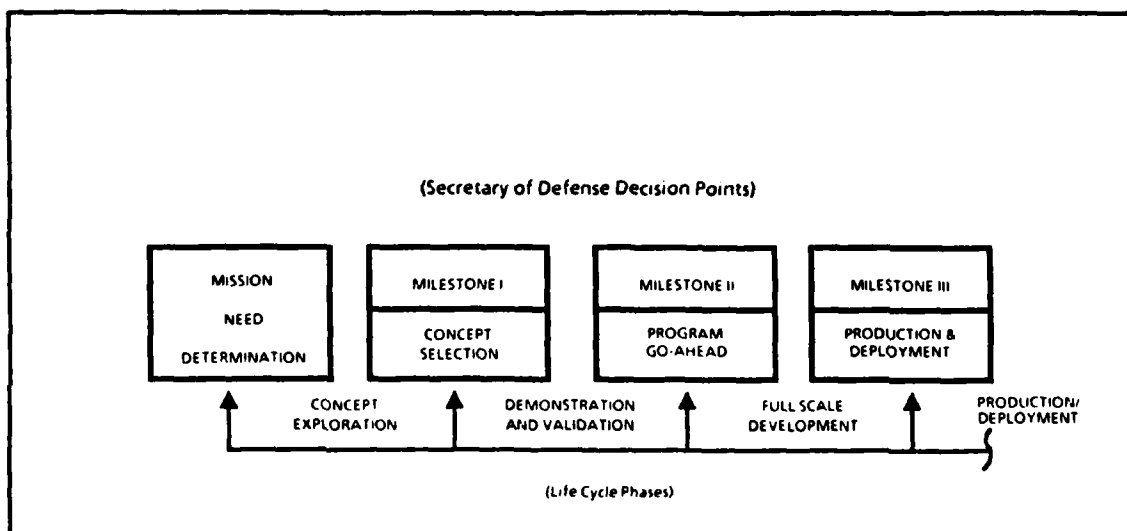


Figure 2.1-1 System Acquisition Life-Cycle Phases and Decision Points

III. The Secretary of Defense, advised by the Defense Acquisition Executive (DAE), decides at these points whether to continue the program and proceed to the next phase or to terminate the program. The system acquisition life cycle applies to the whole system, not the individual parts.

Concept exploration is the initial planning phase, during which the role of and plans for using computer resources in the system are explored. During demonstration and validation, translating operational requirements into functional, interface, and performance requirements is completed; and requirements for each hardware and software configuration item are defined. During FSD, the system is designed, built, tested, and evaluated. These initial three phases should result in a system meeting specified requirements. Production and deployment includes production (if applicable) and delivery and includes all activities involved in supporting the system until it is retired.

2.1.2 Software Development Cycle

The software development cycle, as defined in DOD-STD-SDS, consists of six phases: software requirements analysis, preliminary design, detailed design, coding and unit testing, computer software component (CSC) integration and testing, and computer software configuration item (CSCI) level testing (see Fig. 2.1-2). This cycle, however, is not standardized and there are many variations throughout the industry. Although names and breakdowns vary, the same process is generally followed.

All software requirements are specified during software requirements analysis. The authenticated software requirements specification (signed off by both the customer and contractor) forms the baseline for preliminary design. During preliminary design, a modular, top-level design is developed from the software requirements. During detailed design, the top-level design is refined to successively lower levels until individual units, which perform single, nondivisible functions, are defined. During coding and unit testing, the designer translates the design approach into code and executes verification tests. During CSC integration and testing, code units are integrated and informal tests are performed on aggregates of integrated units. This cycle concludes with CSCI-level testing, during which formal tests are conducted on the software.

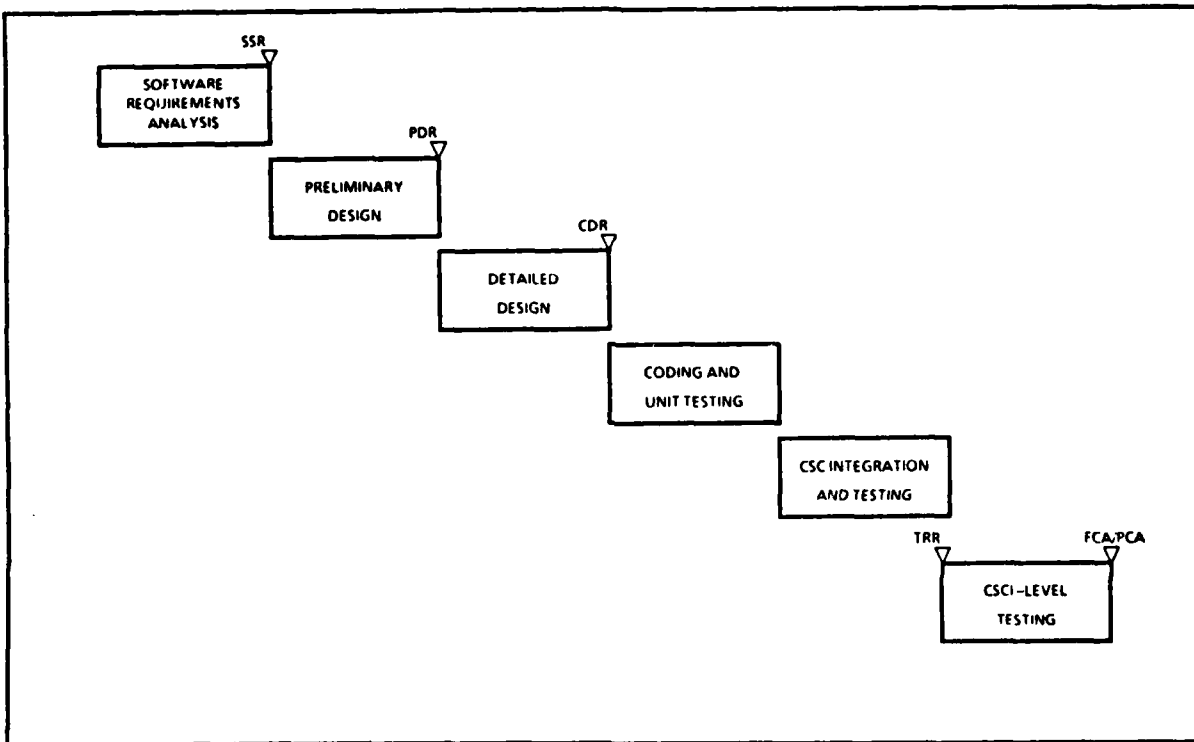


Figure 2.1-2 Software Development Cycle

As with the system acquisition life cycle, the software development cycle has decision points associated with most phases. These decision points (shown in Fig. 2.1-2) are the: software specification review (SSR), preliminary design review (PDR), critical design review (CDR), test readiness review (TRR), and functional configuration audit (FCA)/physical configuration audit (PCA). These decision points are quite different from decision points associated with the system acquisition life cycle. At these decision points it is not determined whether to continue or terminate the program; rather, progress up to that point is reviewed and it is decided if the developer has completed the current phase and is ready to proceed into the next phase.

2.1.3 Life-Cycle Relationships

Each CSCI to be developed goes through the entire software development cycle. The software development cycle can be completed in a single phase of the system acquisition life cycle or can overlap several phases. For example, software could be developed for risk-reduction analysis during concept exploration or demonstration and validation. This software could be used to validate the feasibility of an algorithm or to compare alternative approaches. This type of software may not be in the language required for the operational software and may not be targeted for the same computer. However, it still goes through the entire development cycle. The same is true for test software developed to aid in validation of the operational software. Operational software development may overlap several system life-cycle phases; requirements definition for operational software begins early in the system acquisition life cycle, although operational software is not fully developed until FSD. In this guidebook operational software quality is the primary concern; therefore, the relationship of the operational software development cycle to the system acquisition life cycle will be examined.

There is a specific relationship between the operational software development cycle and the system acquisition life cycle in most system procurements (see Fig. 2.1-3). The software requirements analysis phase overlaps part of the demonstration and validation phase and the beginning of FSD. The remaining operational software development phases occur during FSD; i.e., preliminary design through CSCI-level testing of the software development cycle. This relationship is assumed for the remaining discussions.

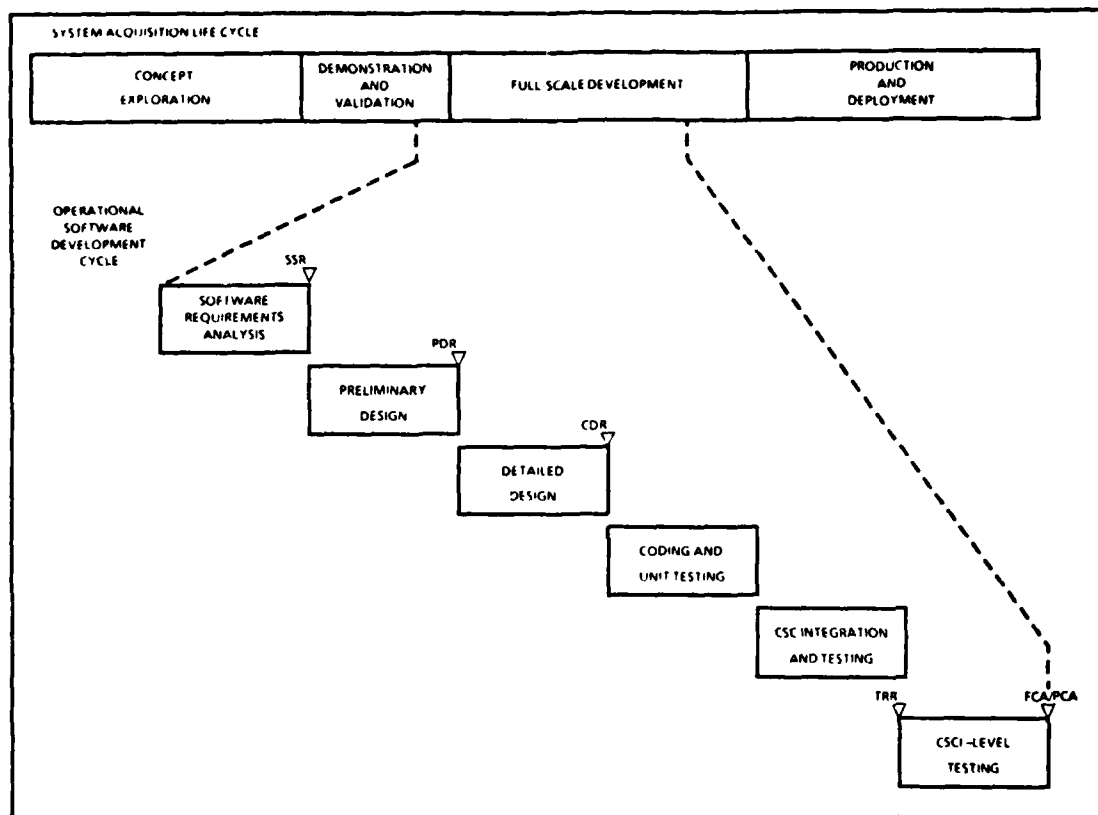


Figure 2.1-3 Life-Cycle Relationship between the System and the Operational Software

2.1.4 Software Acquisition Management

The software acquisition manager has various responsibilities during the software development cycle. This Section focuses on two general functions of software acquisition management: (1) specifying requirements and (2) monitoring development to ensure satisfying the requirements. To describe all that this manager does during the software life cycle is beyond the scope of this guidebook.

Specification of software requirements begins with development of the system specification and continues until all requirements for each CSCI have been specified during software requirements analysis in the software development cycle. These requirements include more than traditional functional and performance requirements. They also include interface, human engineering, language, data base, delivery, self-test, anomaly management, resource reserves, and quality requirements. Many decisions are made to specify these requirements.

The software acquisition manager becomes involved at the system level, when system functional tasks are allocated to software or to hardware. Allocation decisions may be based on trade studies, system engineering, and risk analyses. Once the allocation of functional tasks is completed, specific software requirements can be identified. The result is a set of software capabilities, performance levels, and design constraints. Identification of these specific requirements usually involves decisions supported by trade studies. Such trade studies may include, for example, higher order language (HOL) versus assembly language, distributed processing versus centralized processing, growth capability required for timing and sizing, the degree of human operator interaction required, and efficiency versus maintainability. These software trade studies consider life-cycle costs, risk, schedule, capabilities, software performance, and final product quality. These activities are concluded when the System Program Office (SPO) authenticates (signs off) the software requirements specifications for each CSCI.

Once software requirements are specified, the acquisition manager begins monitoring software development. Monitoring continues throughout preliminary design, detailed design, coding and unit testing, CSC integration and testing, and CSCI-level testing and may continue into the system integration and testing that follows. The primary

concern of monitoring, other than schedule or cost, is whether the software satisfies the requirements. Monitoring provides the acquisition manager with visibility of the evolving product in order to track technical progress and quality. This visibility is achieved through various reviews, audits, documentation, and products required periodically throughout development. Established criteria and measurement methods for each review and audit and for all documentation and products are necessary for tracking progress. Tracking enables the manager to identify problems early enough to correct them. Two activities providing feedback are V&V and QA.

2.1.5 Verification and Validation

The purpose of V&V is to provide the Air Force with systematic assurance that acquired software will perform missions in accordance with requirements. The terms verification and validation are often used interchangeably, but in the software development cycle distinct concepts are associated with each. The meaning of these terms as used here is as follows:

Verification is the iterative process of determining whether the product of each software development phase fulfills requirements levied by the previous phase. That is, (1) software requirements are verified to ensure that they fulfill system-level requirements, (2) the software design is verified to ensure that it satisfies requirements in the software requirements specification, and (3) code is verified to ensure that it complies with the top-level design and detailed design documents. This process does not consider whether system-level software requirements are correct or whether they actually satisfy users needs.

Validation is a continuing process to ensure that requirements at various levels are correct, thus satisfying mission requirements defined by the using command. Sometimes validation is considered to be the system-level test activity that validates the CSCI against software and system requirements. In reality, it is much more than that. Validation, like verification, continues throughout the software life cycle. For example, when software requirements are allocated and derived, a system-level requirement could be found to be vague or incorrect; or during design, it could be discovered that a software requirement is infeasible or ambiguous. Feedback to the

manager enables corrective action to be taken early in development, thereby reducing risk and cost.

The concept of V&V and its relationship to software development products is shown in Figure 2.1-4. V&V provides feedback to the software acquisition manager concerning software technical performance. The term IV&V is used when V&V is done for the Air Force by a contractor other than either the prime contractor or the subcontractor who is developing the software.

2.1.6 Quality Assurance

According to MIL-S-52779A, the purpose of software QA is to ensure that the software delivered under a contract complies with contract requirements. This type of QA program will not ensure development of a high-quality software product unless software quality attributes are specified in measurable terms as part of the contract. The objective of current QA programs is to provide feedback to the acquisition manager concerning various aspects of the development process. QA is similar to V&V, the major difference being that V&V provides technical feedback on software products at only a few points in time, whereas QA provides feedback on a wide range of development activities. But contractual software quality is not normally defined in quantitative terms. The current goal is simply to achieve better quality through controlling the development processes.

Section 2.3 explores how QM technology can help to expand the scope of QA programs to include specification of software quality requirements and measurement of achieved quality levels for software development products. The following paragraphs explain the current scope of QA programs.

At one time, software QA was equated to testing. As an illustration, Section 4 of the CPCI development specification (according to MIL-STD-483) was called Quality Assurance Provisions. However, as with other products, it was learned that quality cannot be tested into software. Because of cost and schedule impacts, it is usually too late to make changes when quality problems are found during testing. Quality can be affected by how code is written and how software is designed. If a software quality problem is found during testing, it is usually very expensive to redesign and to change

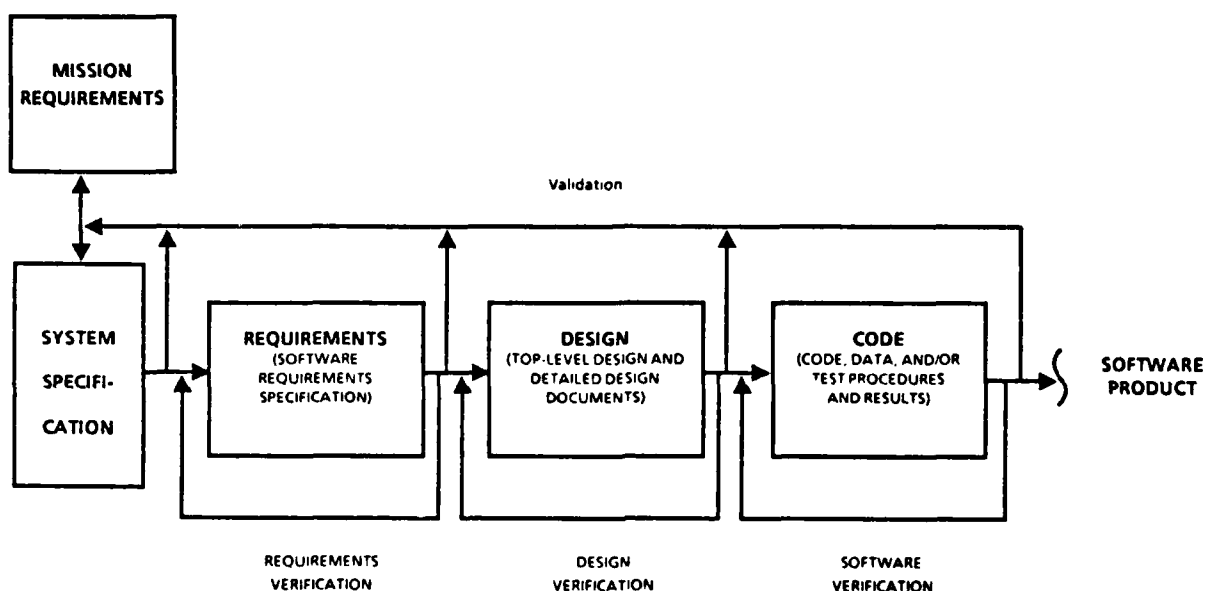


Figure 2.1-4 Relationship of Software Development and V&V

the code. Quality should be planned, designed, and built into software. This realization has led to the current life-cycle-oriented QA approach. This approach focuses attention on all phases of the software development cycle; and software QA now includes many activities, such as ensuring that software is being developed in accordance with plans, that requirements are traceable, that design and code are easily and economically supportable, and that testing is accomplished as planned. These activities provide necessary feedback to the software acquisition manager.

Software quality assurance programs, however, are primarily administrative rather than technical. For example, the QA organization does not trace requirements but ensures that Engineering has developed traceability matrices. The QA function is essentially a checkoff function applied during the software development process; i.e., QA ensures that everything is done as planned. Software QA continues throughout the software development cycle (see Fig. 2.1-5).

Software QA is an evolving discipline. Experience has provided insight into which development practices tend to produce a higher quality software product, and the QA program ensures that selected practices are used by checking the development process. The next step to improving quality is to quantitatively specify quality requirements and to measure and control the quality of the software product as it evolves. Implementing QM technology in the Air Force acquisition process will provide the added dimension of quantitative measures to addressing quality concerns for software products.

2.2 QUALITY METRICS

The purpose of QM technology is to enable the software acquisition manager to specify a desired software quality level for each quality factor of importance to the application and to quantitatively measure the achieved levels of quality at specific points during development. These periodic measurements enable an assessment of current status and a prediction of quality level for the final product. Some problems with delivered software products have been that these products are (to varying degrees) unreliable, incorrect, and/or unmaintainable. QM technology addresses these and other quality-oriented problems by providing a means to specify quality

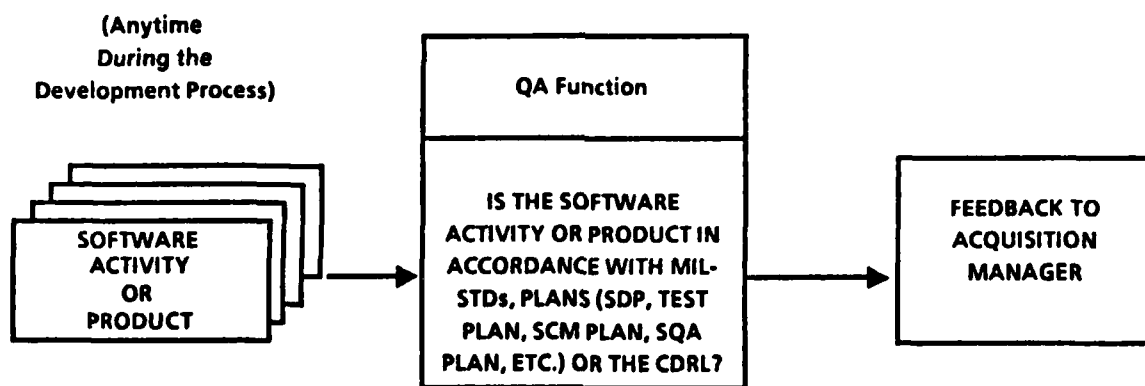


Figure 2.1-5 Software QA Function

requirements, to quantitatively measure quality achieved during development, and to predict a quality level for the final product.

QM technology measures the degree of software quality, not the level of software technical performance; e.g., how easy is it to maintain the software, not how accurate is the navigation algorithm. However, the process of specifying and measuring quality levels is analogous to the process of specifying and measuring technical performance. Both processes begin with similar activities: system needs are assessed, trades are performed (involving resources and levels of performance or levels of quality), and requirements are specified. Subsequent phases involve evaluations of how well these requirements are being satisfied.

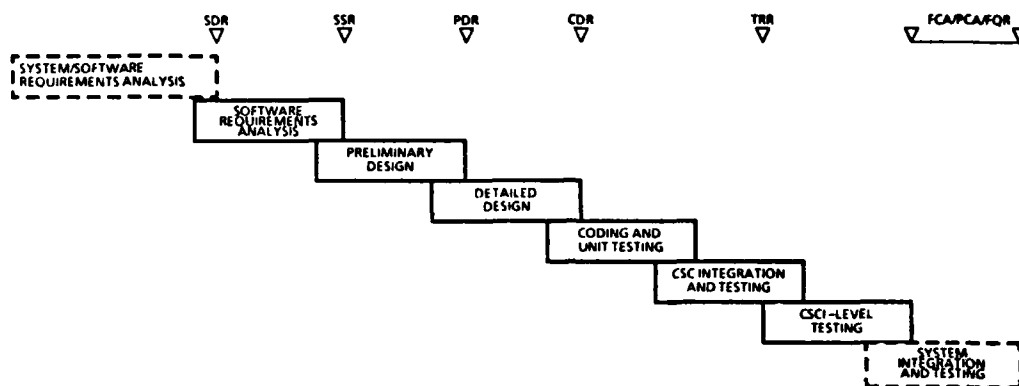
Technical performance levels are traditionally evaluated by modeling in early development stages and by testing in later development stages. Quality has traditionally been evaluated by such methods as reviews, walkthroughs, and audits. This type of quality evaluation ensures that, for example, designs are traceable to requirements, configuration management is adequate, and standards and plans are being followed. However, it does not address such quality issues as software reliability, correctness, and maintainability. QM technology enables a quantitative assessment of these types of quality factors at different stages of development, thereby ensuring that specified quality levels are being satisfied in a manner similar to performance evaluation by testing.

Figure 2.2-1 depicts the software life-cycle model used in QM technology. The software model is shown in typical relationship to two system acquisition phases. Eight development states are shown with typical review and audit points. There are two system-level activities involving software: system/software requirements analysis and system integration and testing (both shown in dashed boxes). (Operational testing and evaluation is the last FSD phase but is not shown as it is not normally performed by the development contractor.) There are six software development phases: software requirements analysis, preliminary design, detailed design, coding and unit testing, CSC integration and testing, and CSCI-level testing. These phases refer to the same development activities as are described in Section 2.1. This division of activities was chosen because at the end of each activity shown in Figure 2.2-1 a configuration baseline generally is established, and software products (specifications, documents,

SYSTEM ACQUISITION PHASES:



Quality Metrics - Software Life Cycle Model:



Quality Metrics - Specification:



Quality Metrics - Monitoring:

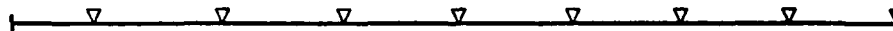


Figure 2.2-1 Quality Metrics Technology - Life-Cycle Model

code) describing that baseline are available for review or audit and the application of metric measurements. Also illustrated in Figure 2.2-1 are the two points at which quality requirements are specified and the eight points at which quality levels are measured (monitored). These measurement points generally correspond to the review or audit points for configuration baselines.

2.2.1 Framework

A hierarchical model for quality has been established (see Fig. 2.2-2). User-oriented factors (e.g., reliability, correctness, maintainability) are at the top level, software-oriented criteria are at the next level, and metrics—quantitative measures of characteristics—are at the lowest level.

This model is flexible in that it indicates a general relationship between each factor and its attributes. This permits updating of individual elements to reflect technology advances without affecting the model itself. For example, as new user concerns evolve, new factors can be added at the top level; and as software technology evolves, criteria and metrics can be added, deleted, or modified as necessary. There are currently 13 quality factors, 29 criteria, 73 metrics, and more than 300 metric elements (distinct parts of a metric). Table 2.2-1 shows the 13 quality factors and describes the primary user concern for choosing each factor. Quality factors and user concerns are categorized by three types of acquisition concerns with respect to the software: (1) product performance—how well does the software function in its normal environment; (2) product design—how valid (appropriate) is the design with respect to requirements, verification, and maintenance; and (3) product adaptation—how easy is it to adapt the software for use beyond its original intended use (e.g., for new requirements, a new application, or a different environment).

Figures 2.2-3, 2.2-4, and 2.2-5 show the quality factors, criteria, and metrics in the hierarchical relationships of the software quality model. The metrics are identified by acronym only in the Figures. These and other framework elements for QM technology are described in detail in Section 3.0. The following sections describe some aspects involved in specifying and monitoring software quality using QM technology.

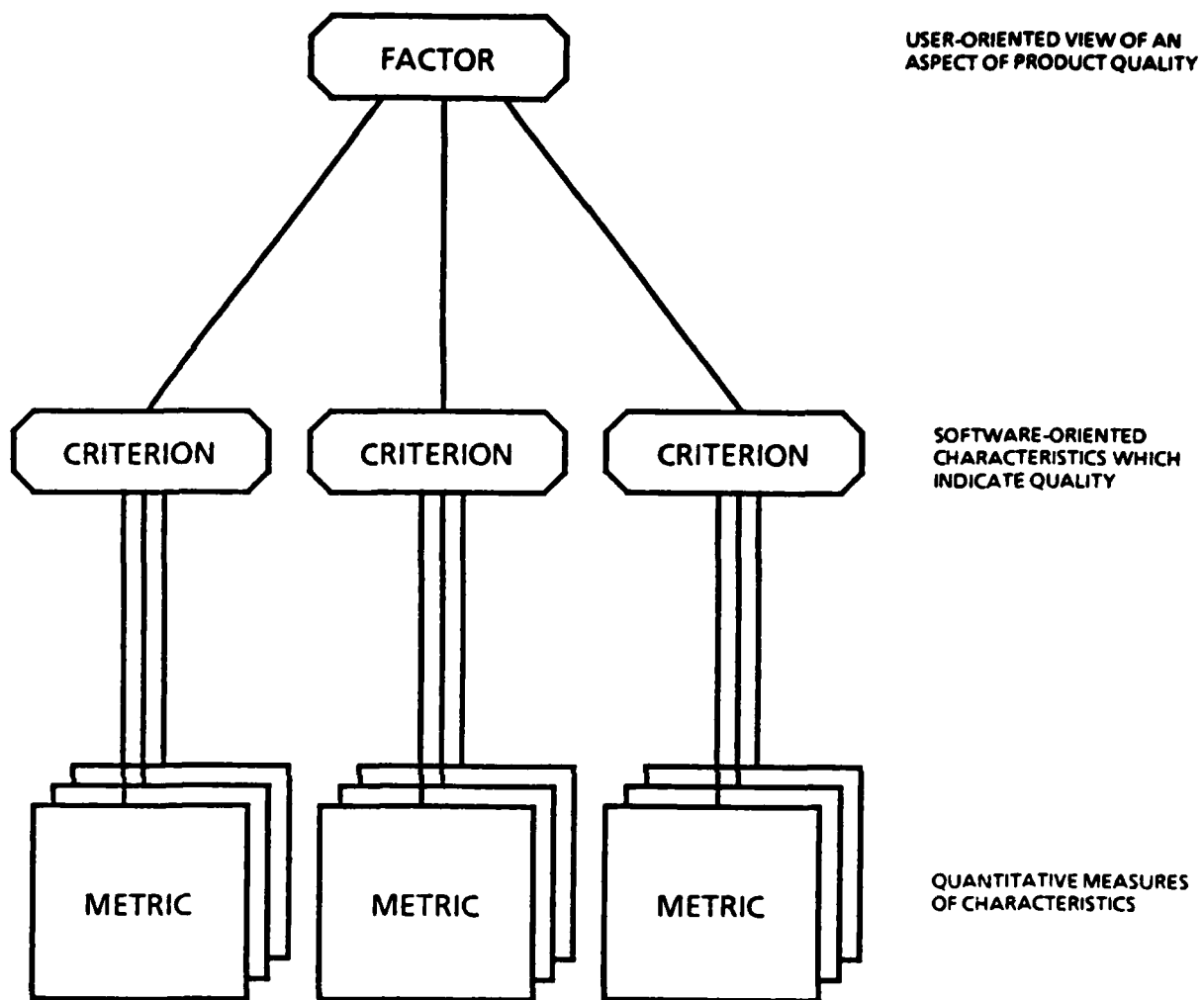


Figure 2.2-2 Software Quality Model

Table 2.2-1 Quality Concerns

Acquisition Concern	User Concern	Quality Factor
PERFORMANCE - HOW WELL DOES IT FUNCTION?	<p>HOW WELL DOES IT UTILIZE A RESOURCE?</p> <p>HOW SECURE IS IT?</p> <p>WHAT CONFIDENCE CAN BE PLACED IN WHAT IT DOES?</p> <p>HOW WELL WILL IT PERFORM UNDER ADVERSE CONDITIONS?</p> <p>HOW EASY IS IT TO USE?</p>	<p>EFFICIENCY</p> <p>INTEGRITY</p> <p>RELIABILITY</p> <p>SURVIVABILITY</p> <p>USABILITY</p>
DESIGN - HOW VALID IS THE DESIGN?	<p>HOW WELL DOES IT CONFORM TO THE REQUIREMENTS?</p> <p>HOW EASY IS IT TO REPAIR?</p> <p>HOW EASY IS IT TO VERIFY ITS PERFORMANCE?</p>	<p>CORRECTNESS</p> <p>MAINTAINABILITY</p> <p>VERIFIABILITY</p>
ADAPTATION - HOW ADAPTABLE IS IT?	<p>HOW EASY IS IT TO EXPAND OR UPGRADE ITS CAPABILITY OR PERFORMANCE?</p> <p>HOW EASY IS IT TO CHANGE?</p> <p>HOW EASY IS IT TO INTERFACE WITH ANOTHER SYSTEM?</p> <p>HOW EASY IS IT TO TRANSPORT?</p> <p>HOW EASY IS IT TO CONVERT FOR USE IN ANOTHER APPLICATION?</p>	<p>EXPANDABILITY</p> <p>FLEXIBILITY</p> <p>INTEROPERABILITY</p> <p>PORTABILITY</p> <p>REUSABILITY</p>

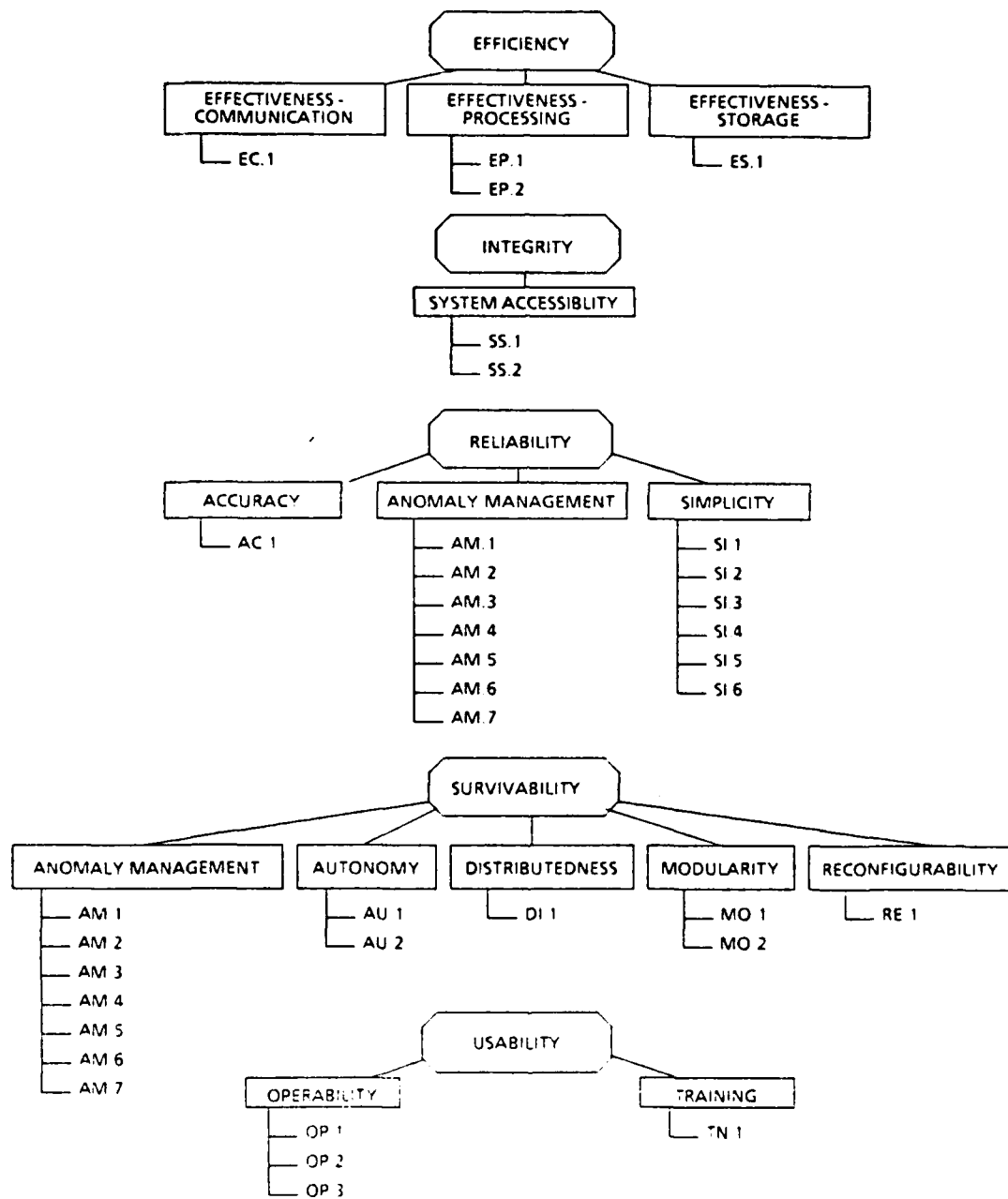


Figure 2.2-3 Performance Factor Attributes

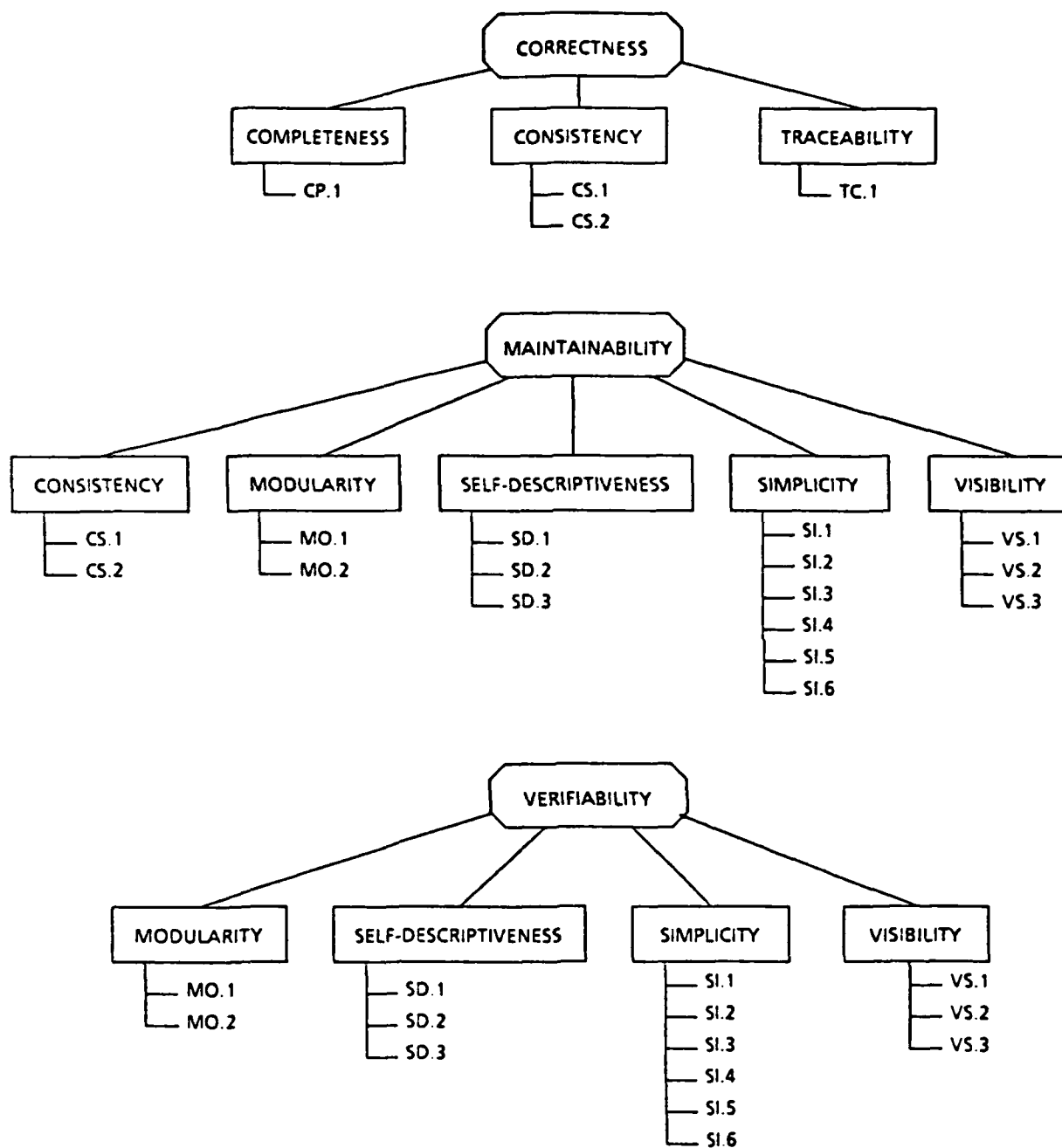


Figure 2.2-4 Design Factor Attributes

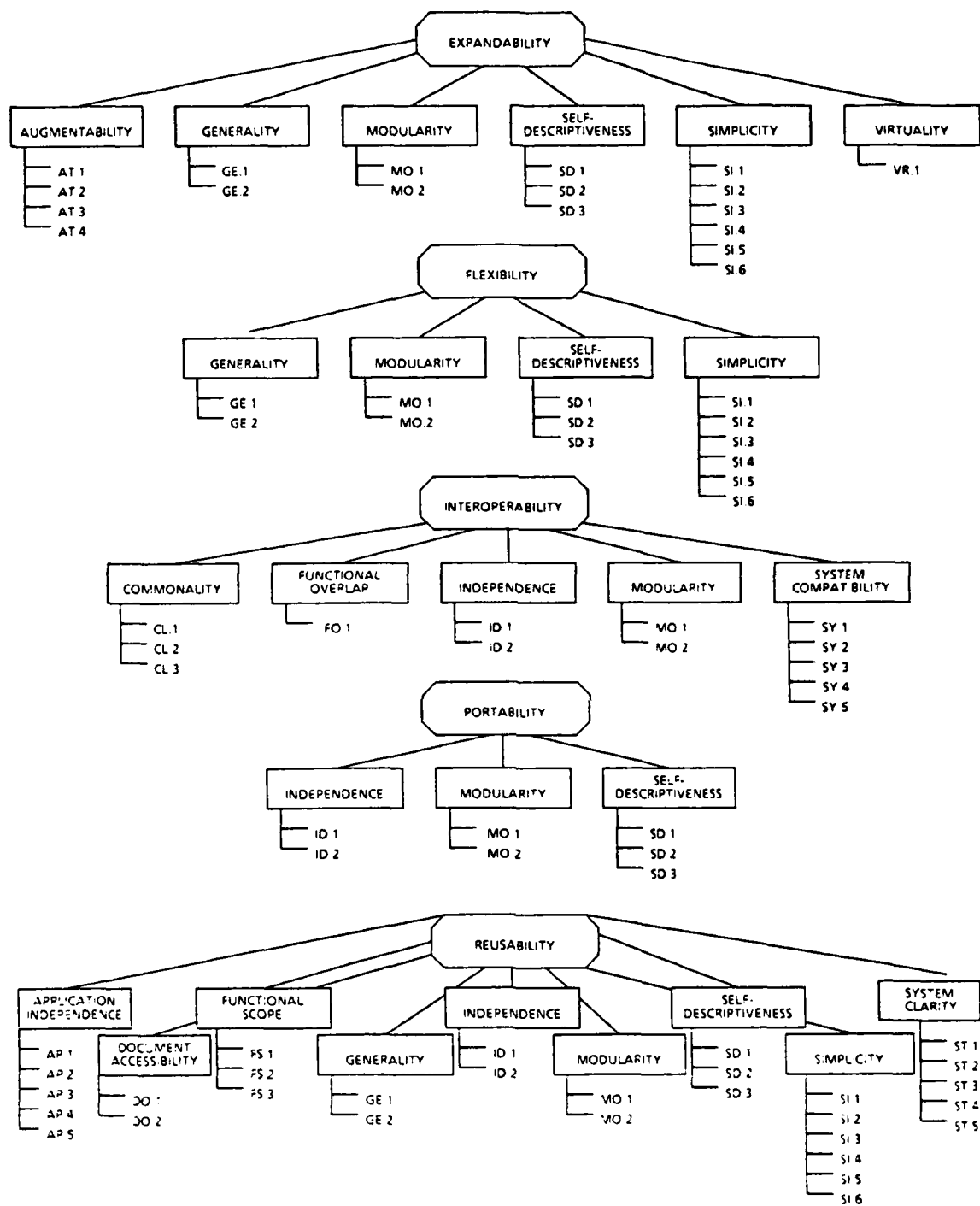


Figure 2.2-5 Adaptation Factor Attributes

2.2.2 Quality Specification

When determining and specifying software quality requirements, system needs are assessed from a quality perspective; the desired quality factors, associated criteria, and applicable metrics are selected; and quality-level goals are derived for each separate quality factor. When assessing system needs, application characteristics should be considered. For example, if the system will have a long life cycle, emphases on maintainability, flexibility, portability, and expandability are recommended. Factor goals define the required quality levels to be achieved for the factor (i.e., excellent, good, or average). In general, choosing a higher quality goal will result in more resources being expended to achieve that level. When deriving factor goals, interrelationships between factors should be considered because a high quality goal for one factor may conflict with a high quality goal for another factor. Table 2.2-2 shows the beneficial and adverse relationships between quality factors; some factors have a positive relationship and others conflict. For example, specifying a high quality level for most factors conflicts with specifying a high quality level for efficiency. These relationships are explored in detail in Section 4.1.3.

A typical problem for an embedded software system arises when reliability is of the utmost importance because of the type of mission to be performed, but efficiency is also required because of space and weight limitations, and flexibility is needed because of the variety of missions and/or targets. It is normally infeasible to select and achieve high quality levels for all three factors. Highly efficient code is usually tightly written assembly-level code and tends to be not as reliable or as amenable to changes (flexible) as looser, more structured HOL code. And code written to be reliable and flexible tends to be less efficient. Trade studies are needed to resolve these problems. If some efficiency is sacrificed for reliability, then performance goals (e.g., for accuracy or range) may be affected. If some flexibility is sacrificed for efficiency, then the scope of the missions and/or targets may be reduced. QM technology provides an aid for decision making when selecting quality-level goals, when determining feasible software requirements, and for allocating acquisition resources. Several iterations of quality tradeoffs may be required for choosing reasonable quality goals. Section 4.0 provides specific techniques for choosing quality factors and includes consideration of application characteristics and factor interrelationships.

Table 2.2-2 Software Quality Factor Interrelationships

ACQUISITION CONCERN	ACQUISITION CONCERN		PERFORMANCE					DESIGN			ADAPTATION				
	QUALITY FACTOR SPECIFIED	QUALITY FACTOR AFFECTED	EFFICIENCY	INTEGRITY	RELIABILITY	SURVIVABILITY	USABILITY	CORRECTNESS	MAINTAINABILITY	VERIFIABILITY	EXPANDABILITY	FLEXIBILITY	INTEROPERABILITY	PORTABILITY	REUSABILITY
PERFORMANCE	EFFICIENCY		▽						△	△				▽	
	INTEGRITY		▽	▽											
	RELIABILITY		▽		▽		△								
	SURVIVABILITY		▽	▽		▽	△		△		▽	▽		▽	▽
	USABILITY		▽				▽		△	△					
DESIGN	CORRECTNESS							▽	△	△	△	△			△
	MAINTAINABILITY		▽						▽	△	△	△			△
	VERIFIABILITY		▽							▽					
ADAPTATION	EXPANDABILITY		▽	▽	▽	▽					▽		△		
	FLEXIBILITY		▽	▽	▽	▽						▽	△		
	INTEROPERABILITY		▽	▽									▽		
	PORTABILITY		▽											▽	
	REUSABILITY		▽	▽	▽	▽			△				△	△	▽

△ = POSITIVE EFFECT

▽ = NEGATIVE EFFECT

BLANK = NONE OR APPLICATION
DEPENDENT

2.2.3 Quality Monitoring

When monitoring software quality, the quality metrics (in the form of questions on worksheets) are applied to software products (specifications, documents, code) at different stages of the development cycle, and a quality-level score is calculated for each factor. The factor score predicts a quality level for the final product. The points in the development cycle where data gathering and analysis are recommended is shown in Figure 2.2-1. These points generally correspond to normal reviews and audits conducted when a configuration baseline has been established (SDR, SSR, PDR, CDR, TRR, and FCA/PCA). Before each review or audit, the metrics selected for the project are applied to software products resulting from that phase of development. This results in a quantitative value for each metric. The metric values are then used to calculate scores for each criterion, and the criteria scores are used to calculate a score (predicted quality level) for each factor.

The quality metrics are applied at incremental points during the development phases. This enables periodic review of progress in meeting quality goal requirements and aids in pinpointing areas of weakness (and strength) in product quality as the product evolves. There are two types of metrics—*anomaly detecting* and *predictive*. Both are used in scoring. A low score for predictive metrics indicates that a low score will probably result for the end product because the design is not considering aspects important to achieving the desired quality level. For example, if the design has very little spare storage capacity, the end product will not be highly expandable. A low score for anomaly-detecting metrics indicates an actual design or code deficiency. For example, if provisions are not made for immediate indication of an access violation, software integrity would be jeopardized. Evaluating low metric scores provides an opportunity for identifying deficiencies and anomalies during development when they are more easily corrected.

Worksheets have been devised to help gather metric data. There is a separate worksheet for each development phase, and each worksheet lists only metrics applicable to that phase. A more detailed explanation of the worksheets is provided in Section 3.4.

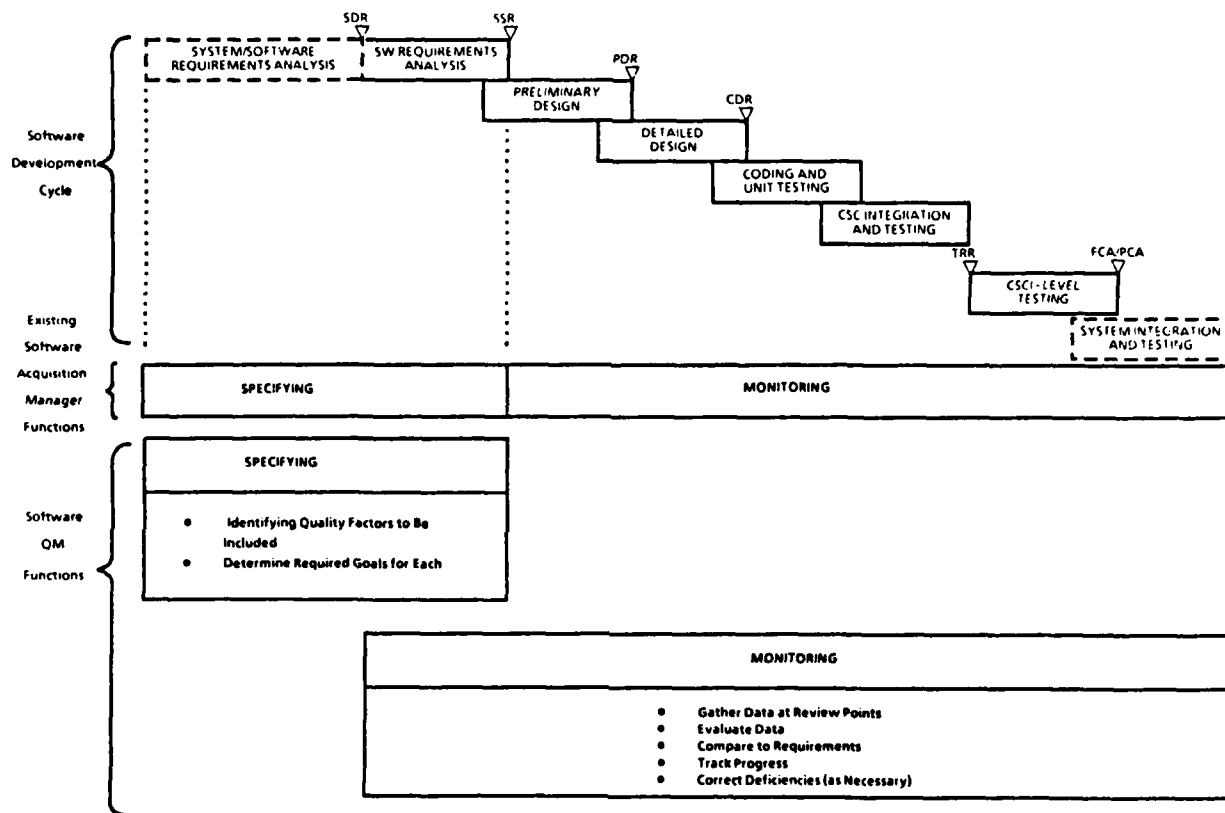


Figure 2.3-1 Software Acquisition Quality Metrics Functions

2.3 SOFTWARE ACQUISITION USING QUALITY METRICS

Two general functions of the software acquisition manager are described in Section 2.1.4: (1) specifying requirements and (2) monitoring development to ensure that requirements are being satisfied. Also two general functions associated with QM technology are described in Sections 2.2.2 and 2.2.3: (1) specifying quality requirements and (2) monitoring development to ensure that metric scores are predicting specified quality goals. When using QM technology, monitoring begins earlier in the development cycle. The relationship of these functions to the software life cycle is shown in Figure 2.3-1.

Specifying and monitoring have not usually overlapped. The specification of software requirements was normally completed before development monitoring began, as shown in Figure 2.3-1. Metric questions have been devised to enable evaluation of software quality reflected in the system specification available at the system design review (SDR). This moves the start of monitoring forward so that the two functions overlap.

Several organizations normally are involved in performing these two functions. Although the internal structure of the Air Force product divisions (ESD, ASD, and SD) may differ, the relationship of the SPO to external organizations is basically the same for each division. Organizations that may be involved in the QM functions and their recommended relationships are shown in Figure 2.3-2. Organizational relationships are discussed in the following paragraphs.

Several organizations should be involved in the specification function. The primary organization responsible for software requirements specification is SPO Software Engineering. However, SPO software engineers need help from both the using command and Air Force Logistics Command (AFLC) to fully define software quality needs. Both organizations have a vested interest in requirements affecting system operation and support.

The using command is primarily interested in operational requirements and is especially qualified to contribute to a definition of quality needs for the performance quality factors (e.g., efficiency, integrity, and reliability). AFLC is primarily interested in support requirements and is especially qualified to contribute to a

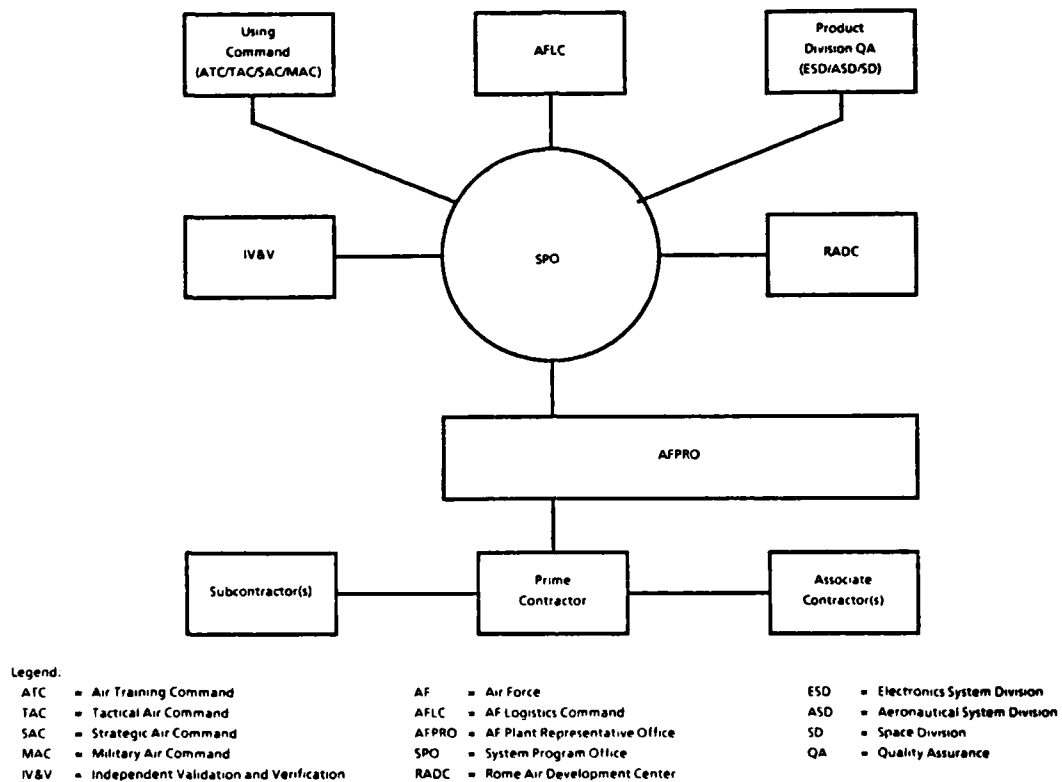


Figure 2.3-2 Air Force Acquisition Relationships Involved in Quality Metrics Functions

definition of quality needs for the design and adaptation quality factors (e.g., maintainability, expandability, and portability). With input from these organizations, SPO Software Engineering can determine the contractual statement of quality requirements. In addition, the Product Division Software QA organization is normally tasked to ensure that quality requirements are included in the contract. These responsibilities and relationships for the specification function are shown in Figure 2.3-3.

Several organizations also should be involved in the monitoring function. Among the first activities are identifying and negotiating with the organization that will collect and analyze metric data. If that organization is to be another Air Force agency, such as Air Force Contracts Management Division (AFCMD), then the SPO needs to negotiate the effort through a memorandum of agreement. If the organization is to be an IV&V contractor, then the IV&V contract needs to be negotiated. These negotiations must be completed very early in the program before data collection starts, and SPO Software Engineering must ensure that necessary support is provided.

Several organizations could collect and analyze data, including SPO Software Engineering, the Product Division Software QA, the Air Force Plant Representative Office (AFPRO), and an IV&V contractor. The following criteria were established to aid in selecting an organization: technical capability, labor availability, economy, and data availability. Technical capability refers to the depth of technical understanding of software by people in the organization. Labor availability refers to availability of qualified people to perform this additional task (i.e., currently available or readily obtainable). Economy refers to the least costly method for the SPO to obtain data. Data availability refers to the ability to access the most current contractor documentation and information. Informal lines of communication greatly influence this factor.

We rated four candidate organizations using these criteria, based on our experience. A score of 1 represents the best conditions and a 3 represents the worst for each criterion. A total unweighted score was determined for each organization, with the lowest score representing the best choice. The evaluation scores are shown in Table 2.3-1.

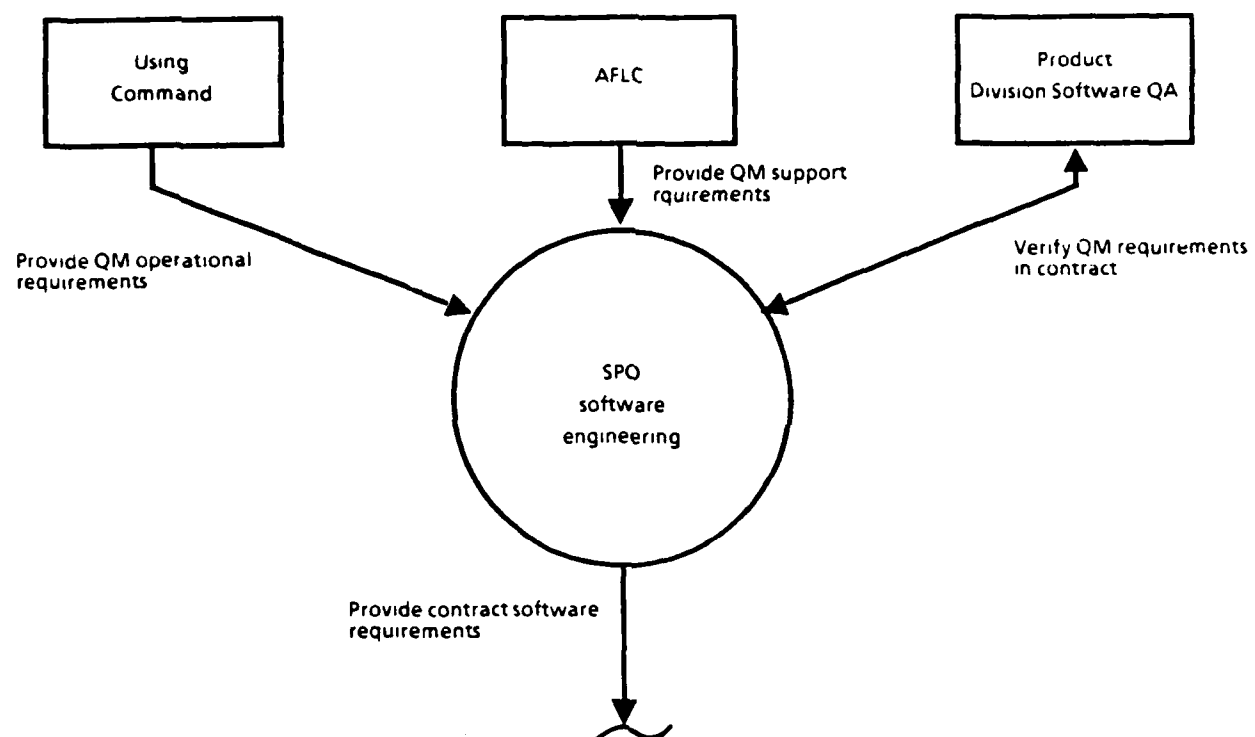


Figure 2.3-3

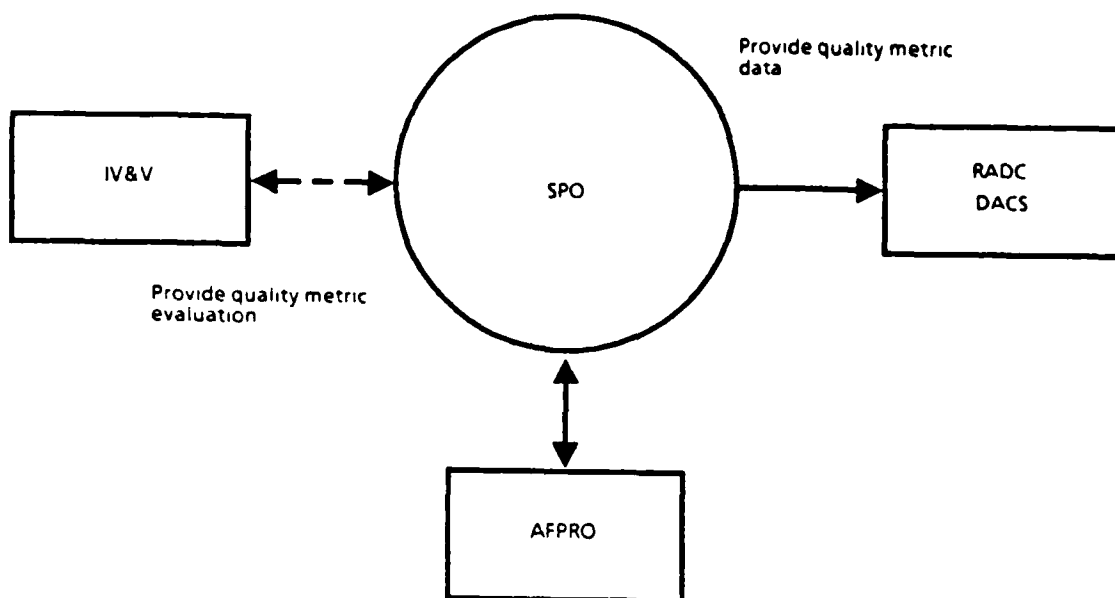
Recommended Responsibilities and Relationships for the QM Specification Function

Table 2.3-1 Organizational Evaluation

<div>CRITERION</div> <div>ORGANIZATION</div>	TECHNICAL CAPABILITY	LABOR AVAILABILITY	ECONOMY	DATA AVAILABILITY	SCORE* SUMMARY
SPO ENGINEERING	2	2	1	2	7
PRODUCT DIVISION SOFTWARE QA	3	3	1	3	10
AFPRO	2	2	1	1	6
IV&V	1	1	3	2	7

1 = BEST
2 = MEDIUM
3 = WORST

* Lowest Score is Best (Unweighted)



--- Indicates alternate source

Figure 2.3-4

Recommended Responsibilities and Relationships for the QM Monitoring Function

Several assumptions were made for scoring. The first was that all criteria are weighted equally; actually, however, technical capability and labor availability may be overriding factors for selection. For technical capability, it was assumed that Product Division Software QA groups are unlikely to be able to obtain people experienced in both software engineering and QA to perform that job. For economy, it was assumed that any Air Force person (civilian or military) is a free resource for the SPO. Otherwise, the SPO must pay for IV&V contractor services. Data availability scores include the assumption that the IV&V contractor works for SPO Software Engineering and that good communication channels are established. These assumptions may not be valid in all situations.

The AFPRO received the lowest score and, therefore, was rated best. It is generally recommended that the AFPRO perform data collection and analysis for the SPO. When this cannot be negotiated, it is recommended that an IV&V contractor be assigned this task. Although SPO Software Engineering and the IV&V contractor are rated equally, the recommendation to use an IV&V contractor was made because of better labor availability. It is recommended that a chart similar to the one shown in Table 2.3-1 be developed early in a program.

A proposed DID, Software Quality Evaluation Report, is contained in Appendix C and can be used to report data collection and analysis results to the software acquisition manager. This feedback enables the manager to track progress, ensure that requirements are being satisfied, and take corrective action when necessary. Recommendations for responsible organizations and relationships for monitoring are shown in Figure 2.3-4. We recommend that the Data and Analysis Center for Software (DACS) at Rome be used as the data base for quality metrics information and that the SPO provide a copy of the quality requirements and all metric data to DACS (e.g., provide a copy of the Software Quality Evaluation Report). This has the advantages of providing one centralized location for all QM data and enabling access to all historical data by any one product division. It also enables large-scale data analysis and correlation to be performed on data from all product divisions. Any changes in QM technology such as new factors, metrics, and worksheet formats should be disseminated from a central point. This concept is illustrated in Figure 2.3-5.

Framework Elements:

- Factors
- Criteria
- Metrics
- Metric Elements
- Worksheets
- Scoresheets

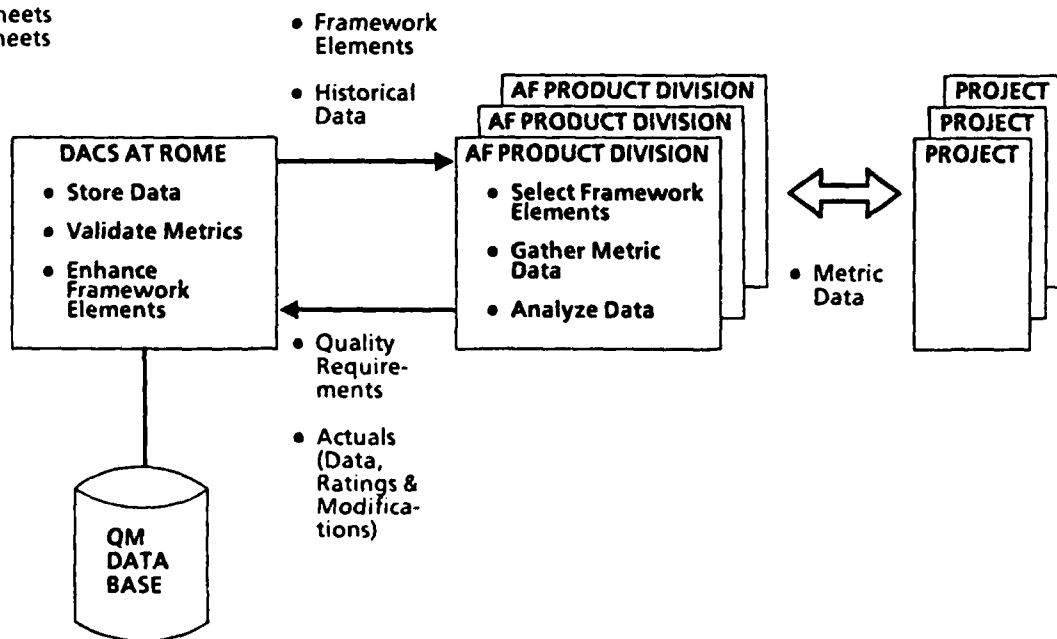


Figure 2.3-5 Relationship between Product Divisions and DACS

The preceding paragraphs discuss government monitoring only, and the development contractor was not mentioned. Because quality factor requirements are included as contractual requirements, the development contractors must also monitor achieved quality levels to show compliance. However, to ensure that data and reports received by the SPO are unbiased, we recommend that the government independently monitor achieved quality levels.

2.4 POTENTIAL BENEFITS AND PROBLEMS

This section discusses the potential benefits and problems associated with integrating QM technology into the software acquisition management process and of using QM technology during acquisition.

2.4.1 Benefits

Possible benefits of using QM technology include a higher quality end product, greater emphasis on quality throughout the life cycle, better management control, and life-cycle cost savings. A high-quality end product is possible because required quality levels are specified quantitatively. There is little room for misinterpretation or for undesirable results such as a highly efficient but unreliable and unmaintainable product. The acquisition manager is assured that the end product is of the required degree of quality. Also, other software requirements are considered at the same time that quality requirements are being specified. This means that the quality requirements should be reasonable and should not conflict with functional and performance requirements (or vice versa), thereby increasing the likelihood that all software requirements can be satisfied within allocated resources. In addition, achieved quality levels are monitored throughout development providing increased visibility for control of quality. Periodic application of metrics provides the acquisition manager with adequate feedback about software development progress and enables early redirection if necessary. Finally, evaluating specific low metric scores provides an additional mechanism for detecting deficiencies and anomalies in requirements, design, and code.

Life-cycle cost savings are possible for several reasons. Using metrics to detect deficiencies and anomalies enables correction during development. Correction at this

time is less costly than during operation and maintenance. Also, it is possible to be more precise about funding for quality. If adequate quality levels are achieved during development, it is unnecessary to spend more effort in raising quality levels or in developing a near-perfect product.

The greatest cost savings potential comes from having certain qualities actually built into the software. For example, if system A has a high level of reusability built into the software, then cost savings result from building system B reusing a portion of system A software. These potential cost savings are available for other quality factors such as flexibility, portability, interoperability, and expandability. Details for considering cost are described in Section 4.0.

Other benefits can also be realized. For example, use of QM technology can provide the acquisition manager an added assurance that the required degree of reliability is achieved in the final product. This would be especially important in acquisitions involving space applications or nuclear armaments.

2.4.2 Problems

There are potential technical and administrative problems when using quality metrics in acquisitions; i.e., in integrating QM technology into the Air Force software acquisition process. Problems could arise during one of the most important tasks, that of maintaining a current QM technology baseline. Baseline changes could result from, for example, changes in quality factor ratings, new factor ratings being established, new metrics being established, and metrics being validated for new application areas. Changes could originate from any product division using QM technology. Using DACS would minimize the risk of such problems as: multiple baselines in the product divisions, duplication of validation efforts, and use of outdated information (e.g., outdated ratings).

A potential problem could arise where subjective judgment is required in scoring some metrics. Two people gathering metric data from the same software products could score the worksheets differently. This risk has been minimized by rewriting the questions on the metric worksheets so that they are clear, simple, and understandable. Also, metric element explanations have been included for clarification. As more

historical information becomes available, it will be possible to do a reasonableness check on worksheet data entries, based on previous data ranges. However, we recommend that experienced personnel perform data collection and that education and training be provided for personnel involved with QM technology.

Another potential problem might arise when attempting to automate portions of the data gathering task through an automated measuring tool. This type of tool scans source code and outputs statistics on the code (e.g., percentage of comments, number of specific constructs). The scanner is language dependent and must be developed for each language, but standardization on a language (e.g., Ada) will minimize cost.

Problems with organizational structures and manpower may be encountered when implementing QM technology at the product divisions. Program offices do not have QA divisions. QA in the program office is usually done by Engineering. In addition, software QA organizations in the product divisions are relatively new. These organizations are trying to define their role in the acquisition process and their relationship to the program offices. Absence of a well-defined organizational structure for software QA could lead to disagreements over assigning QM responsibilities. Either organization could resist accepting responsibility for QM functions because of staffing problems. Program offices are usually not fully staffed with software engineers; to accept more responsibilities without additional personnel would be difficult. Software QA organizations have small staffs and find it difficult to hire qualified personnel. A person with experience in both software engineering and QA is required, but few software engineers are interested in QA assignments. Staffing problems should receive attention during implementation of QM technology in the Air Force software acquisition process.

3.0 QUALITY METRICS FRAMEWORK

This section describes elements of the software quality framework. Terminology and concepts introduced in this Section are used throughout subsequent Sections.

The goals of quality metrics (QM) technology are to enable a software acquisition manager to (1) specify the types and degrees of software qualities desired in the end product and (2) predict end-product quality levels through measuring the degree of those qualities present during development. The Rome Air Development Center (RADC) quality program (see Sec. 1.1) has established a model for viewing software quality. Figure 2.2-2 depicts this model, showing a hierarchical relationship between a quality factor, criteria, and metrics. Criteria and metrics are factor attributes.

Quality factors (e.g., reliability, usability, correctness, and maintainability) are user-oriented terms, each representing an aspect of software quality. Thirteen quality factors are used to specify the types of qualities wanted in a particular software product. Product environment and expected use affect emphasis. For example, if human lives could be affected, integrity, reliability, correctness, verifiability, and survivability would be emphasized. If the software is expected to have a long life cycle, maintainability and expandability would be emphasized.

Criteria are software-oriented terms representing software characteristics. For example, operability and training are criteria for usability. The degree to which these characteristics are present in the software is an indication of the degree of presence of an aspect of quality (i.e., a quality factor).

Metrics are software-oriented details of a characteristic (a criterion) of the software. Each metric is defined by a number of metric elements. The metric elements enable quantification of the degree of presence of criteria and, hence, factors. "Are all the errors specified which are to be reported to the operator/user?" is an example metric element question for the criterion operability (see worksheet 0, OP.1(2), App. A).

Using the methodology described in Section 4.0, the acquisition manager is responsible for specifying needed quality factors by priority, with quality levels commensurate

Table 3.1-1 Software Quality Factor Definitions and Rating Formulas

ACQUISITION CONCERN	QUALITY FACTOR	DEFINITION	RATING FORMULA
PERFORMANCE	EFFICIENCY	RELATIVE EXTENT TO WHICH A RESOURCE IS UTILIZED (i.e. STORAGE SPACE, PROCESSING TIME, COMMUNICATION TIME)	1- $\frac{\text{ACTUAL RESOURCE UTILIZATION}}{\text{ALLOCATED RESOURCE UTILIZATION}}$
	INTEGRITY	EXTENT TO WHICH THE SOFTWARE WILL PERFORM WITHOUT FAILURES DUE TO UNAUTHORIZED ACCESS TO THE CODE OR DATA WITHIN A SPECIFIED TIME PERIOD	1- $\frac{\text{ERRORS}}{\text{LINES OF CODE}}$
	RELIABILITY	EXTENT TO WHICH THE SOFTWARE WILL PERFORM WITHOUT ANY FAILURES WITHIN A SPECIFIED TIME PERIOD	1- $\frac{\text{ERRORS}}{\text{LINES OF CODE}}$
	SURVIVABILITY	EXTENT TO WHICH THE SOFTWARE WILL PERFORM AND SUPPORT CRITICAL FUNCTIONS WITHOUT FAILURES WITHIN A SPECIFIED TIME PERIOD WHEN A PORTION OF THE SYSTEM IS INOPERABLE	1- $\frac{\text{ERRORS}}{\text{LINES OF CODE}}$
	USABILITY	RELATIVE EFFORT FOR USING SOFTWARE (TRAINING AND OPERATION) (e.g. FAMILIARIZATION, INPUT PREPARATION, EXECUTION, OUTPUT INTERPRETATION)	1- $\frac{\text{LABOR DAYS TO USE}}{\text{LABOR YEARS TO DEVELOP}}$
DESIGN	CORRECTNESS	EXTENT TO WHICH THE SOFTWARE CONFORMS TO ITS SPECIFICATIONS AND STANDARDS	1- $\frac{\text{ERRORS}}{\text{LINES OF CODE}}$
	MAINTAINABILITY	EASE OF EFFORT FOR LOCATING AND FIXING A SOFTWARE FAILURE WITHIN A SPECIFIED TIME PERIOD	1- 0.1 (AVERAGE LABOR-DAYS TO FIX)
	VERIFIABILITY	RELATIVE EFFORT TO VERIFY THE SPECIFIED SOFTWARE OPERATION AND PERFORMANCE	1- $\frac{\text{EFFORT TO VERIFY}}{\text{EFFORT TO DEVELOP}}$
ADAPTATION	EXPANDABILITY	RELATIVE EFFORT TO INCREASE THE SOFTWARE CAPABILITY OR PERFORMANCE BY ENHANCING CURRENT FUNCTIONS OR BY ADDING NEW FUNCTIONS OR DATA	1- $\frac{\text{EFFORT TO EXPAND}}{\text{EFFORT TO DEVELOP}}$
	FLEXIBILITY	EASE OF EFFORT FOR CHANGING THE SOFTWARE IN FUNCTIONS, OR DATA TO SATISFY OTHER REQUIREMENTS	1- 0.05 (AVERAGE LABOR-DAYS TO CHANGE)
	INTEROPERABILITY	RELATIVE EFFORT TO COUPLE THE SOFTWARE OF ONE SYSTEM TO THE SOFTWARE OF ANOTHER SYSTEM	1- $\frac{\text{EFFORT TO COUPLE}}{\text{EFFORT TO DEVELOP}}$
	PORTABILITY	RELATIVE EFFORT TO TRANSPORT THE SOFTWARE FOR USE IN ANOTHER ENVIRONMENT (HARDWARE CONFIGURATION AND/OR SOFTWARE SYSTEM ENVIRONMENT)	1- $\frac{\text{EFFORT TO TRANSPORT}}{\text{EFFORT TO DEVELOP}}$
	REUSABILITY	RELATIVE EFFORT TO CONVERT A SOFTWARE COMPONENT FOR USE IN ANOTHER APPLICATION	1- $\frac{\text{EFFORT TO CONVERT}}{\text{EFFORT TO DEVELOP}}$

NOTE: THE RATING VALUE RANGE IS FROM 0 TO 1. IF THE VALUE IS LESS THAN 0, THE RATING VALUE IS ASSIGNED TO 0.

with cost consideration. Factor requirements are provided as part of the software requirements (along with operational, performance, and design requirements). This enables the corresponding criteria and metrics to be identified and used to measure the degree of presence of desired qualities at key review points during development, allowing periodic predictions of the quality level for the final product. Metric worksheets and scoresheets help in applying the metrics and in determining metric scores.

3.1 SOFTWARE QUALITY FACTORS

Thirteen software quality factors are identified in Table 2.2-1, with the user concern that characterizes the need for each type of quality. Quality factors are shown grouped under one of three acquisition concerns: performance, design, or adaptation. An acquisition manager specifying requirements for software will likely do so in a DOD-STD-SDS format in four main areas: (1) software performance characteristics (performance), (2) software design and construction (design), (3) anticipated software expansion or reuse (adaptation), and (4) quality assurance (including quality metrics). The similarity of areas and acquisition concerns enables the acquisition manager to easily identify and select quality factor categories and specific factors of interest. Quality criteria are similarly categorized (see Sec. 3.2); thus, selecting criteria and metrics is simplified.

3.1.1 Factor Definitions and Rating Formulas

Quality factor definitions and factor rating formulas are shown in Table 3.1-1. Rating formulas quantify user concerns for the final product. The formulas use three types of measurements: (1) number of errors per lines of code (2) effort to perform an action and (3) utilization of resources. Ratings should fall in the range from zero to one. The rating formula for reliability is one minus the number of errors per lines of code. For example, if one error per 1,000 lines of code occur during a given time period (e.g., during operational testing and evaluation) the rating formula shows a reliability level of 0.999 ($1 - 1/1,000 = 0.999$).

During software development, metrics are applied to software products, and a metric score is calculated for the appropriate factors. This metric score is an estimation (or

APPLICATION PHASE ACQUISITION CONCERN/ QUALITY FACTOR	INITIAL USE OF PRODUCT			NEW USE OF PRODUCT		
	SOFTWARE DEVELOPMENT	OPERATIONAL TESTING AND EVALUATION	PRODUCTION AND DEPLOYMENT	SOFTWARE DEVELOPMENT	OPERATIONAL TESTING AND EVALUATION	PRODUCTION AND DEPLOYMENT
PERFORMANCE				<div> <div></div> <div></div> <div></div> <div></div> <div></div> </div> <div>SAME AS FOR INITIAL USE, AS REQUIRED</div>		
EFFICIENCY						
INTEGRITY						
RELIABILITY						
SURVIVABILITY						
USABILITY						
DESIGN				<div> <div></div> <div></div> <div></div> </div> <div>SAME AS FOR INITIAL USE, AS REQUIRED</div>		
CORRECTNESS						
MAINTAINABILITY						
VERIFIABILITY						
ADAPTATION						
EXPANDABILITY						
FLEXIBILITY						
INTEROPERABILITY		(AR)	(AR)			
PORTABILITY						
REUSABILITY						

= RATING ESTIMATION = RATING ASSESSMENT (AR) = AS REQUIRED

Figure 3.1-1 Rating Estimation and Rating Assessment Windows

prediction) of what the quality level will be for the final product. Figure 3.1-1 indicates the timeframes during which rating values are estimated through metric scores (closed box) and the timeframes during which rating values can be assessed by using actual data and the rating formula (dotted box). For example, the rating value for reliability is estimated by using metric scores during software development. During operational testing and evaluation and during production and deployment, actual data on number of errors per lines of code become available to assess the rating and evaluate predictions made during development. Exact correlations between metric scores and rating values have not been established. Research has only shown that higher metric scores during development result in higher quality end products. Table 3.1-2 shows a range of values for each rating formula that might occur when using actual data (e.g., during production and deployment) to assess rating values. The values shown are hypothetical.

The following paragraphs describe the factors and rating formulas in each acquisition concern category.

Performance. Performance quality factors deal both with the ability of the software to function and with error occurrences that affect software functioning. Low quality levels predict poor software performance. These quality factors are efficiency, integrity, reliability, survivability, and usability.

Efficiency deals with utilization of a resource. The rating formula for efficiency is in terms of actual utilization of a resource and budgeted allocation for utilization. For example, if a unit is budgeted for 10% available memory and actually uses 7%, the rating formula shows an efficiency level of 0.3 ($1 - 0.07/0.10 = 0.3$).

Integrity deals with software security failures due to unauthorized access. The rating formula for integrity is in terms of number of integrity-related software errors occurring during a given time (e.g., during operational testing and evaluation) and total number of executable lines of source code. This formula is similar to the formula for reliability; the difference is that reliability is concerned with all software errors, and integrity is concerned only with the subset of errors that affect integrity. For example, if three integrity-related errors per 10,000 lines of code occurred during operational testing and evaluation, the rating formula shows an integrity level of 0.9997 ($1 - 1/10,000 = 0.9997$).

Table 3.1-2 Quality Factor Ratings

Quality factor	Rating formula	Rating information			
Efficiency	1- $\frac{\text{Actual utilization}}{\text{Allocated utilization}}$	Value	0.1	0.3	0.5
		% utilization	90%	70%	50%
Integrity	1- $\frac{\text{Errors}}{\text{Lines of code}}$	Value	0.9995	0.9997	0.9999
		Errors/LOC	5/10,000	3/10,000	1/10,000
Reliability	1- $\frac{\text{Errors}}{\text{Lines of code}}$	Value	0.995	0.997	0.999
		Errors/LOC	5/1,000	3/1,000	1/1,000
Survivability	1- $\frac{\text{Errors}}{\text{Lines of code}}$	Value	0.9995	0.9997	0.9999
		Errors/LOC	5/10,000	3/10,000	1/10,000
Usability	1- $\frac{\text{Labor-days to use}}{\text{Labor-years to develop}}$	Value	0.5	0.7	0.9
		Days/years	5/10	6/20	10/100
Correctness	1- $\frac{\text{Errors}}{\text{Lines of code}}$	Value	0.9995	0.9997	0.9999
		Errors/LOC	5/10,000	3/10,000	1/10,000
Maintainability	1- 0.1 (average labor-days to fix)	Value	0.8	0.9	0.95
		Average labor-days	2.0	1.0	0.5
Verifiability	1- $\frac{\text{Effort to verify}}{\text{Effort to develop}}$	Value	0.4	0.5	0.6
		% effort	60%	50%	40%
Expandability	1- $\frac{\text{Effort to expand}}{\text{Effort to develop}}$	Value	0.8	0.9	0.95
		% effort	20%	10%	5%
Flexibility	1- 0.05 (average labor-days to change)	Value	0.8	0.9	0.95
		Average labor-days	40	20	10
Interoperability	1- $\frac{\text{Effort to couple}}{\text{Effort to develop}}$	Value	0.9	0.95	0.99
		% effort	10	5	1
Portability	1- $\frac{\text{Effort to transport}}{\text{Effort to develop}}$	Value	0.9	0.95	0.99
		% effort	10	5	1
Reusability	1- $\frac{\text{Effort to convert}}{\text{Effort to develop}}$	Value	0.4	0.6	0.8
		% effort	60	40	20

Reliability concerns any software failure. The rating formula for reliability is in terms of total number of software errors occurring during a specified time and total number of executable lines of source code. For example, if three errors per 1,000 lines of code occurred during operational testing and evaluation, the rating formula shows a reliability level of 0.997 ($1 - 3/1,000 = 0.997$).

The concern with survivability is that software continue to perform (e.g., in a degraded mode) even when a portion of the system has failed. The rating formula for survivability is in terms of number of survivability-related errors (the subset of errors that affect survivability) occurring during a specified time and total number of executable lines of source code. This formula is similar to the formula for reliability.

Usability deals with relative effort involved in learning about and using software. The rating formula for usability is in terms of average effort to use software (to train for using it and to operate it) and original development effort. This formula considers size of the software system in rating usability. It is recommended that effort to use be expressed in labor-days and effort for original development be expressed in labor-years to maintain a scoring range consistent with that of other factors. For example, if 10 labor-days were required for training on a system that required 100 labor-years to develop, the rating formula shows a usability level of 0.9 ($1 - 10/100 = 0.9$); and if five labor-days were required for training on a system that required 10 labor-years to develop, the rating formula shows a usability level of 0.5 ($1 - 5/10 = 0.5$).

Design. Design quality factors deal mainly with software failure and correction. Low quality levels usually result in repeating a portion of the development process (e.g., redesign, recode, reverify); hence the term design. The factors are correctness, maintainability, and verifiability.

Correctness deals with the extent to which software design and implementation conform to specifications and standards. Criteria of correctness (completeness, consistency, and traceability) deal exclusively with design and documentation formats. Under the three criteria there are no metrics dealing with content material affecting software operation or performance. The rating formula for correctness is in terms of number of specifications-related and standards-related errors that occur after formal

release of the specifications and standards and total number of executable lines of source code. This formula is also similar to the formula for reliability; the difference is that correctness is concerned only with that subset of errors related to violations of specified requirements and nonconformance to standards.

Maintainability is concerned with ease of effort in locating and fixing software failures. The rating formula for maintainability is in terms of average number of labor-days to locate and fix an error within a specified time (e.g., during production and deployment). For example, if an average of 0.5 labor-days were required to locate and fix errors during production and deployment, the rating formula shows a maintainability level of 0.95 ($1 - (0.1 \times 0.5) = 0.95$).

Verifiability deals with software design characteristics affecting the effort to verify software operation and performance. The rating formula for verifiability is in terms of effort to verify software operation and performance and original development effort. This formula is similar to the adaptation, effort-ratio formulas. For example, if 40% of the development effort is spent reviewing and testing software, the rating formula shows a verifiability level of 0.6 ($1 - 0.40/1.00 = 0.6$).

Adaptation. These quality factors deal mainly with using software beyond its original requirements, such as extending or expanding capabilities and adapting for use in another application or in a new environment. Low quality levels predict relatively high costs for new software use. Quality factors are expandability, flexibility, interoperability, portability, and reusability.

Expandability deals with relative effort in increasing software capabilities or performance. The rating formula for expandability is in terms of effort to increase software capability and performance and original development effort. For example, if five labor-months were spent enhancing software performance for software that originally took 100 labor-months to develop, the rating formula shows an expandability level of 0.95 ($1 - 5/100 = 0.95$).

Flexibility deals with ease of effort in changing software to accommodate changes in requirements. The rating formula for flexibility is in terms of average effort to change software to satisfy other (i.e., new or modified) requirements within a

specified time. For example, if an average of one labor-day was required to modify software functioning during operational testing and evaluation, the rating formula shows a flexibility level of 0.95 ($1 - (0.05 \times 1) = 0.95$).

Interoperability is concerned with relative effort in coupling software of one system to software of one or more other systems. The rating formula for interoperability is in terms of effort to couple and original development effort and is similar to the formula for expandability.

Portability deals with relative effort involved in transporting software to another environment (e.g., different host processor, operating system, executive). The rating formula for portability is in terms of effort to transport software for use in another environment and original development effort and is similar to the formula for expandability.

Reusability is concerned with relative effort for converting a portion of software for use in another application. The rating formula for reusability is in terms of effort to convert software for use in another application and original development effort and is similar to the formula for expandability.

If adaptation effort is greater than original development effort, the effort-ratio formulas will yield a quality level value less than zero. In this case, the quality level value is assigned to zero. (This situation is considered unlikely because it would probably be less expensive to develop a new product than to adapt an existing one.)

3.1.2 Quality Factor Interrelationships

Relationships exist among quality factors; some relationships are synergistic and others conflicting. Specifying requirements for more than one type of quality for a product can possibly have either a beneficial or an adverse effect on cost to provide the quality. Factor relationships and relative cost to provide are discussed in Section 4.0.

Table 3.2-1 Software Quality Factors and Criteria

	ACQUISITION CONCERN		PERFORMANCE					DESIGN			ADAPTATION				
	FACTOR/ACRONYM	CRITERION/ACRONYM	EFFICIENCY	INTEGRITY	RELIABILITY	SURVIVABILITY	USABILITY	CORRECTNESS	MAINTAINABILITY	VERIFIABILITY	EXPANDABILITY	FLEXIBILITY	INTEROPERABILITY	PORTABILITY	REUSABILITY
ACQUISITION CONCERN			E	I	R	S	U	C	M	V	E	F	I	P	R
PERFORMANCE	ACCURACY	AC			X										
	ANOMALY MANAGEMENT	AM			X	X									
	AUTONOMY	AU				X									
	DISTRIBUTEDNESS	DI				X									
	EFFECTIVENESS - COMMUNICATION	EC	X												
	EFFECTIVENESS - PROCESSING	EP	X												
	EFFECTIVENESS - STORAGE	ES	X												
	OPERABILITY	OP					X								
	RECONFIGURABILITY	RE				X									
DESIGN	SYSTEM ACCESSIBILITY	SS													
	TRAINING	TN		X			X								
	COMPLETENESS	CP						X							
	CONSISTENCY	CS						X	X						
ADAPTATION	TRACEABILITY	TC						X							
	VISIBILITY	VS							X	X					
	APPLICATION INDEPENDENCE	AP													X
	AUGMENTABILITY	AT									X				
	COMMONALITY	CL											X		X
	DOCUMENT ACCESSIBILITY	DO													
	FUNCTIONAL OVERLAP	FO											X		
	FUNCTIONAL SCOPE	FS													X
	GENERALITY	GE									X	X			X
GENERAL	INDEPENDENCE	ID											X	X	X
	SYSTEM CLARITY	ST											X		X
	SYSTEM COMPATIBILITY	SY											X		X
	VIRTUALITY	VR									X				
	MODULARITY	MO				X			X	X	X	X	X	X	X
	SELF-DESCRIPTIVENESS	SD							X	X	X	X		X	X
	SIMPLICITY	SI			X				X	X	X	X			X

3.2 SOFTWARE QUALITY CRITERIA

Criteria are software-oriented terms representing software characteristics. Software quality criteria can be grouped under the same three acquisition concerns as quality factors: performance, design, and adaptation. Table 3.2-1 shows the relationship of criteria to quality factors. Four categories for criteria are shown: performance, design, adaptation, and general. Each criterion is an attribute of one or more quality factors. The criteria in the first three categories are solely attributes of factors within the same acquisition concern (i.e., performance, design, and adaptation). Criteria in the fourth category are factor attributes within more than one acquisition concern.

Criteria and factors within each category are listed alphabetically for easy referencing. Alphabetizing by name or by acronym gives the same sequence. Criteria definitions are listed in Table 3.2-2.

3.3 SOFTWARE QUALITY METRICS

Metrics are software-oriented details of a software characteristic (a criterion). Each criterion consists of one or more metrics. Each metric is an attribute of only one criterion. Table 3.3-1 lists the name and acronym of each criterion (in alphabetical order) and the name and acronym of each metric that is an attribute of that criterion. Metric acronyms are acronym extensions of the parent criterion. For example, the acronym for the criterion commonality is CL; the acronym for the three metric attributes are CL.1, CL.2, and CL.3.

Each metric is defined by one or more metric elements. Metric elements are detailed questions applied to software products; answers to them enable quantification of metrics and of the parent criterion and factor. Metric elements are designated by acronym only (no name) and are listed on the metric worksheets. Acronym designation is an extension of the parent metric acronym. For example, the 14 metric element acronyms for the metric CL.1 are CL.1 (1) through CL.1 (14).

Table 3.2-2 Quality Criteria Definitions

ACQ- UI- SION CON- CERN-	CRITERION	ACRONYM	DEFINITION
P E R F O R M A N C E	ACCURACY	AC	<ul style="list-style-type: none"> Those characteristics of software which provide the required precision in calculations and outputs
	ANOMALY MANAGEMENT	AM	<ul style="list-style-type: none"> Those characteristics of software which provide for continuity of operations under and recovery from non-nominal conditions
	AUTONOMY	AU	<ul style="list-style-type: none"> Those characteristics of software which determine its non-dependency on interfaces and functions
	DISTRIBUTEDNESS	DI	<ul style="list-style-type: none"> Those characteristics of software which determine the degree to which software functions are geographically or logically separated within the system
	EFFECTIVENESS-COMM	EC	<ul style="list-style-type: none"> Those characteristics of the software which provide for minimum utilization of communications resources in performing functions
	EFFECTIVENESS-PROCESSING	EP	<ul style="list-style-type: none"> Those characteristics of the software which provide for minimum utilization of processing resources in performing functions.
	EFFECTIVENESS-STORAGE	ES	<ul style="list-style-type: none"> Those characteristics of the software which provide for minimum utilization of storage resources.
	OPERABILITY	OP	<ul style="list-style-type: none"> Those characteristics of software which determine operations and procedures concerned with operation of software and which provide useful inputs and outputs which can be assimilated
	RECONFIGURABILITY	RE	<ul style="list-style-type: none"> Those characteristics of software which provide for continuity of system operation when one or more processors, storage units, or communication links fails
	SYSTEM ACCESSIBILITY	SS	<ul style="list-style-type: none"> Those characteristics of software which provide for control and audit of access to the software and data
	TRAINING	TN	<ul style="list-style-type: none"> Those characteristics of software which provide transition from current operation and provide initial familiarization
D E S I G N	COMPLETENESS	CP	<ul style="list-style-type: none"> Those characteristics of software which provide full implementation of the functions required
	CONSISTENCY	CS	<ul style="list-style-type: none"> Those characteristics of software which provide for uniform design and implementation techniques and notation
	TRACEABILITY	TC	<ul style="list-style-type: none"> Those characteristics of software which provide a thread of origin from the implementation to the requirements with respect to the specified development envelope and operational environment
	VISIBILITY	VS	<ul style="list-style-type: none"> Those characteristics of software which provide status monitoring of the development and operation
A D A P T A T I O N	APPLICATION INDEPENDENCE	AP	<ul style="list-style-type: none"> Those characteristics of software which determine its nondependency on database system, microcode, computer architecture, and algorithms
	AUGMENTABILITY	AT	<ul style="list-style-type: none"> Those characteristics of software which provide for expansion of capability for functions and data
	COMMONALITY	CL	<ul style="list-style-type: none"> Those characteristics of software which provide for the use of interface standards for protocols, routines, and data representations
	DOCUMENT ACCESSIBILITY	DO	<ul style="list-style-type: none"> Those characteristics of software which provides for easy access to software and selective use of its components
	FUNCTIONAL OVERLAP	FO	<ul style="list-style-type: none"> Those characteristics of software which provide common functions to both systems
	FUNCTIONAL SCOPE	FS	<ul style="list-style-type: none"> Those characteristics of software which provide commonality of functions among applications
	GENERALITY	FE	<ul style="list-style-type: none"> Those characteristics of software which provide breadth to the functions performed with respect to the application
	INDEPENDENCE	ID	<ul style="list-style-type: none"> Those characteristics of software which determine its non-dependency on software environment (computing system, operating system, utilities, input, output routines, libraries)
	SYSTEM CLARITY	ST	<ul style="list-style-type: none"> Those characteristics of software which provide for clear description of program structure in a non-complex and understandable manner
	SYSTEM COMPATIBILITY	SY	<ul style="list-style-type: none"> Those characteristics of software which provide the hardware, software, and communication compatibility of two systems
G E N E R A L	VIRTUALITY	VR	<ul style="list-style-type: none"> Those characteristics of software which present a system that does not require user knowledge of the physical, logical, or topological characteristics
	MODULARITY	MO	<ul style="list-style-type: none"> Those characteristics of software which provide a structure of highly cohesive components with optimum coupling
	SELF-DESCRIPTIVENESS	SD	<ul style="list-style-type: none"> Those characteristics of software which provide explanation of the implementation of functions
	SIMPLICITY	SI	<ul style="list-style-type: none"> Those characteristics of software which provide for definition and implementation of functions in the most noncomplex and understandable manner

Table 3.3-1 Quality Metrics Summary

CRITERION		METRIC	
NAME	ACRONYM	NAME	ACRONYM
ACCURACY	AC	ACCURACY CHECKLIST	AC.1
ANOMALY MANAGEMENT	AM	ERROR TOLERANCE/CONTROL IMPROPER INPUT DATA COMPUTATIONAL FAILURES HARDWARE FAULTS DEVICE ERRORS COMMUNICATIONS ERRORS NODE/COMMUNICATION FAILURES	AM.1 AM.2 AM.3 AM.4 AM.5 AM.6 AM.7
APPLICATION INDEPENDENCE	AP	DATA BASE MANAGEMENT IMPLEMENTATION INDEPENDENCE DATA STRUCTURE ARCHITECTURE STANDARDIZATION MICROCODE INDEPENDENCE FUNCTIONAL INDEPENDENCE	AP.1 AP.2 AP.3 AP.4 AP.5
AUGMENTABILITY	AT	DATA STORAGE EXPANSION COMPUTATION EXTENSIBILITY CHANNEL EXTENSIBILITY DESIGN EXTENSIBILITY	AT.1 AT.2 AT.3 AT.4
AUTONOMY	AU	INTERFACE COMPLEXITY SELF-SUFFICIENCY	AU.1 AU.2
COMMONALITY	CL	COMMUNICATIONS COMMONALITY DATA COMMONALITY COMMON VOCABULARY	CL.1 CL.2 CL.3
COMPLETENESS	CP	COMPLETENESS CHECKLIST	CP.1
CONSISTENCY	CS	PROCEDURE CONSISTENCY DATA CONSISTENCY	CS.1 CS.2
DISTRIBUTEDNESS	DI	DESIGN STRUCTURE	DI.1
DOCUMENT ACCESSIBILITY	DO	ACCESS TO DOCUMENTATION WELL-STRUCTURED DOCUMENTATION	DO.1 DO.2
EFFECTIVENESS-COMMUNICATION	EC	COMMUNICATION EFFECTIVENESS MEASURE	EC.1
EFFECTIVENESS-PROCESSING	EP	PROCESSING EFFECTIVENESS MEASURE DATA USAGE EFFECTIVENESS MEASURE	EP.1 EP.2
EFFECTIVENESS-STORAGE	ES	STORAGE EFFECTIVENESS MEASURE	ES.1
FUNCTIONAL OVERLAP	FO	FUNCTIONAL OVERLAP CHECKLIST	FO.1
FUNCTIONAL SCOPE	FS	FUNCTION SPECIFICITY FUNCTION COMMONALITY FUNCTION SELECTIVE USABILITY	FS.1 FS.2 FS.3
GENERALITY	GE	UNIT REFERENCING UNIT IMPLEMENTATION	GE.1 GE.2
INDEPENDENCE	ID	SOFTWARE INDEPENDENCE FROM SYSTEM MACHINE INDEPENDENCE	ID.1 ID.2
MODULARITY	MO	MODULAR IMPLEMENTATION MODULAR DESIGN	MO.1 MO.2
OPERABILITY	OP	OPERABILITY CHECKLIST USER INPUT COMMUNICATIVENESS USER OUTPUT COMMUNICATIVENESS	OP.1 OP.2 OP.3
RECONFIGURABILITY	RE	RESTRUCTURE CHECKLIST	RE.1
SELF-DESCRIPTIVENESS	SD	QUANTITY OF COMMENTS EFFECTIVENESS OF COMMENTS DESCRIPTIVENESS OF LANGUAGE	SD.1 SD.2 SD.3
SIMPLICITY	SI	DESIGN STRUCTURE STRUCTURED LANGUAGE OR PREPROCESSOR DATA AND CONTROL FLOW COMPLEXITY CODING SIMPLICITY SPECIFICITY HALSTEAD'S LEVEL OF DIFFICULTY MEASURE	SI.1 SI.2 SI.3 SI.4 SI.5 SI.6

Table 3.3-1 Quality Metrics Summary (continued)

CRITERION		METRIC	
NAME	ACRONYM	NAME	ACRONYM
SYSTEM ACCESSIBILITY	SS	ACCESS CONTROL ACCESS AUDIT	SS.1 SS.2
SYSTEM CLARITY	ST	INTERFACE COMPLEXITY PROGRAM FLOW COMPLEXITY APPLICATION FUNCTIONAL COMPLEXITY COMMUNICATION COMPLEXITY STRUCTURE CLARITY	ST.1 ST.2 ST.3 ST.4 ST.5
SYSTEM COMPATIBILITY	SY	COMMUNICATION COMPATIBILITY DATA COMPATIBILITY HARDWARE COMPATIBILITY SOFTWARE COMPATIBILITY DOCUMENTATION FOR OTHER SYSTEM	SY.1 SY.2 SY.3 SY.4 SY.5
TRACEABILITY	TC	CROSS REFERENCE	TC.1
TRAINING	TN	TRAINING CHECKLIST	TN.1
VIRTUALITY	VR	SYSTEM/DATA INDEPENDENCE	VR.1
VISIBILITY	VS	UNIT TESTING INTEGRATION TESTING CSCI TESTING	VS.1 VS.2 VS.3

3.4 METRIC WORKSHEETS

Metric worksheets are contained in Appendix A. The worksheets contain metric elements as questions. Software products (specifications, documents, and source listings) are used as source information to answer questions on worksheets; answers are then translated into metric element scores (yes = 1, no = 0, and a formula answer results in a score from 0 to 1). This enables scoring of the parent metric, criterion, and factor and results in a quality level indication for the product.

Seven different worksheets are applied in different development phases. Table 3.4-1 indicates the timeframe during an acquisition life-cycle phase when a worksheet is used, shows the software level of abstraction at which the worksheet is applied, and lists key terminology used within the worksheet.

Worksheet 0 is applied to products of system/software requirements analysis. The worksheet is applied at the system level. (For large systems, software may not be a discernible component in the design with separate requirements at the system level. In this case, worksheet 0 is applied at the system segment level.)

Worksheet 1 is applied to products of software requirements analysis. A separate worksheet is used for each CSCI.

Worksheet 2 is applied to products of preliminary design. A separate worksheet is used for each CSCI.

Worksheets 3A and 3B are applied to products of detailed design. A separate worksheet 3A is used for each CSCI. A separate worksheet 3B is used for each unit of a CSCI. Worksheets 3A and 3B are applied together; answers on 3B worksheets for CSCI units are used in scoring the 3A worksheet for that CSCI.

Worksheets 4A and 4B are applied to products of code and unit testing. Worksheets 4A and 4B are applied in the same manner as 3A and 3B. A separate worksheet 4A is used for each CSCI, and a separate worksheet 4B is used for each CSCI unit.

Table 3.4-1 Metric Worksheet/Life-Cycle Correlation

Life-Cycle Phase/ Phase/ Activity		Demonstration & Validation		Full-Scale Development (FSD)							
				System/ Software Requirements Analysis	Software Requirements Analysis	Preliminary Design	Detailed Design	Coding & Unit Testing	CSC Integration & Testing	CSCI - Level Testing	System Integration & Testing
Application Level/ Terminology		System	<ul style="list-style-type: none">SystemSystem functionCSCI	Metric Worksheet 0							
CSCI	<ul style="list-style-type: none">CSCISoftware function		Metric Worksheet 1								
CSCI	<ul style="list-style-type: none">CSCITop-level CSC			Metric Worksheet 2							
CSCI	<ul style="list-style-type: none">CSCITop-level CSCLower-level CSCUnit				Metric Worksheet 3A		Metric Worksheet 4A				
UNIT	<ul style="list-style-type: none">Unit				Metric Worksheet 3B		Metric Worksheet 4B				
<div>(Selected metric questions are reapplied during the integration and testing phases as indicated in the quality attribute correlation table in Appendix A.)</div>											

For the remainder of the development cycle, selected metric questions are reapplied as indicated in the quality attribute correlation Table in Appendix A.

Metric worksheets are designed to be applied at specific levels (e.g., CSCI, unit). Worksheets can be applied at other levels; however, some questions may not be applicable. For example, if worksheet 1 were applied to a CSCI function, question CP.1(6) should be deleted or reworded because it only applies at the CSCI level.

Metric worksheets are designed to be applied to software development products identified in DOD-STD-SDS. The minimum product set is listed by software development phase in Table 3.4-2. Each product is identified by title and by DID number. Information from the entire set of products for a particular phase is needed as source material to answer metric questions on the worksheet applicable to that phase. It is not necessary to specify the complete product set for each acquisition, only to have equivalent information available to answer worksheet questions. For example, when acquiring a small system, information regarding the QA plan and software standards may be included as part of the software development plan.

3.5 FACTOR SCORESHEETS

Factor scoresheets are contained in Appendix B. There are 13 factor scoresheets, one for each software quality factor. Scoresheets are used for translating information at the metric element level on the worksheets into a quality level score for a quality factor. Each scoresheet has blanks for the factor and for all attributes of that factor (i.e., criteria, metrics, and metric elements). Worksheet information is transferred to the scoresheets at the metric element level. "Yes" answers are scored as 1; "no" answers are scored as 0; and numeric answers resulting from formulas are transferred directly to scoresheets (scoring range from 0 to 1). Scores are then calculated for the parent metrics, criteria, and factor according to the hierarchical (attribute) relationship indicated on the scoresheet.

Table 3.4-2 Software Development Products

Phase/Product Title	Applicable DID
System/Software Requirements Analysis	
System/Segment Specification	DI-S-X101
Software Development Plan	DI-A-X103
Preliminary Software Requirements Specification	DI-E-X107
Operational Concept Document	DI-M-X125
Software Quality Assurance Plan	DI-R-X105
Software Problem/Change Report	DI-E-X106
Software Standards and Procedures Manual	DI-M-X109
Preliminary Interface Requirements Specification	DI-E-X108
Software Requirements Analysis	
Software Requirements Specification	DI-E-X107
Interface Requirements Specification	DI-E-X108
Preliminary Design	
Software Top-Level Design Document	DI-E-X110
Software Test Plan	DI-T-X116
Preliminary Software User's Manual	DI-M-X121
Preliminary Computer System Operator's Manual	DI-M-X120
Detailed Design	
Software Detailed Design Document	DI-E-X111
Software Test Description	DI-T-X117
Data Base Design Document	DI-E-X113
Interface Design Document	DI-E-X112
Coding and Unit Testing	
Source Code/Listings	(Appendix)
Preliminary Software Test Procedure	DI-T-X118
CSC Integration and Testing	
Software Test Procedure	DI-T-X118
CSCI-Level Testing	
Software Product Specification	DI-E-X114
Software Test Report(s)	DI-T-X119
Software User's Manual	DI-M-X121
Computer System Operator's Manual	DI-M-X120
System Integration and Testing	
Software Product Specification	DI-E-X114
Software Test Report(s)	DI-T-X119
Software User's Manual	DI-M-X121
Computer System Operator's Manual	DI-M-X120

4.0 SOFTWARE QUALITY SPECIFICATION METHODOLOGY

This section describes a methodology for determining and specifying software quality requirements for command and control applications. The methodology includes procedures for determining and specifying quality factor requirements, techniques for making quantifiable tradeoffs among quality factors, techniques for relating quality levels to cost over the software life cycle, and procedures for analyzing quality measurement data.

Methodology Overview. Specifying software quality requirements is part of a larger process for using quality metrics in software acquisition management. Figure 4.0-1 shows this process in two major parts: software quality specification, including assessment of compliance with requirements, and software quality evaluation (measurement of achieved quality levels). This document, the Software Quality Specification Guidebook, provides guidance for specification. The Software Quality Evaluation Guidebook provides guidance for evaluation.

In Section 2.0, two quality metrics functions—specification and monitoring—were described. Specification includes identifying and detailing quality requirements and monitoring includes gathering and reducing data, comparing results with requirements, and taking corrective action if necessary. Section 4.0 groups these functional activities into two slightly different categories—specification and evaluation—to enable separating the guidebooks for personnel who will be performing different functions. Software quality specification, as shown in Figure 4.0-1, includes identifying and specifying requirements and assessing compliance with those requirements since these are the responsibility of System Program Office (SPO) personnel. Results of compliance assessment are used to initiate corrective action. Software quality evaluation includes only data collection and analysis and generation of the Software Quality Evaluation Report since these are the responsibility of the development contractor or an independent verification and validation (IV&V) contractor (or an Air Force organization, as is discussed in Sec. 2.3).

The process begins early in the system life cycle—usually during system demonstration and validation. We assume that a description of the nature of the system and system needs or requirements exists. This description could be a statement of work or a draft

system specification and is the primary basis for identifying software quality factor requirements. A series of procedural steps is performed to determine specific software quality needs and to specify quality requirements. Steps include polling groups such as the Air Force using command and the Air Force Logistics Command (AFLC) in order to provide a comprehensive set of operational and support quality requirements from a quality factor point of view. These steps could be performed by the SPO or the development contractor or through awarding a separate contract.

Software quality requirements are entered into the system requirements specification and are treated as contractual obligations (just the same as technical requirements). As the system contractor proceeds with development, quality requirements from the system requirements specification are allocated to lower level specifications and finally assigned to units within the software detailed design document in a manner similar to that for other requirements. This requirements flow is shown in Figure 4.0-2. Each time during the cycle that development products are released (usually at major review points such as system design review (SDR), software specification review (SSR), preliminary design review (PDR), and critical design review (CDR)), quality metrics, in the form of metric worksheets, are applied to the products. Raw data are then used to calculate scores indicating quality level achieved for each quality factor, and these scores are compared to specified requirements.

Application of metrics and scoring of achieved product quality levels are performed by the development contractor to show compliance with quality requirements. It is anticipated that product evaluation will also be performed in parallel by another group such as an IV&V team, the AFPRO, SPO Software Engineering, or Product Division Software Quality Assurance, as is discussed in Section 2.3. Data collection and analysis results are documented in a Software Quality Evaluation Report (see App. C). This report is reviewed separately at major review points. The report is included in the review package released before the review date. The SPO uses these results to assess compliance with quality requirements and (1) approves or disapproves of compliance variations at the review and/or (2) respecifies quality requirements and ensures that changes are reflected in the system requirements specification.

Use of the Methodology and Guidebooks. The methodology and guidebooks were designed primarily for use on projects during which quality requirements are specified early in the

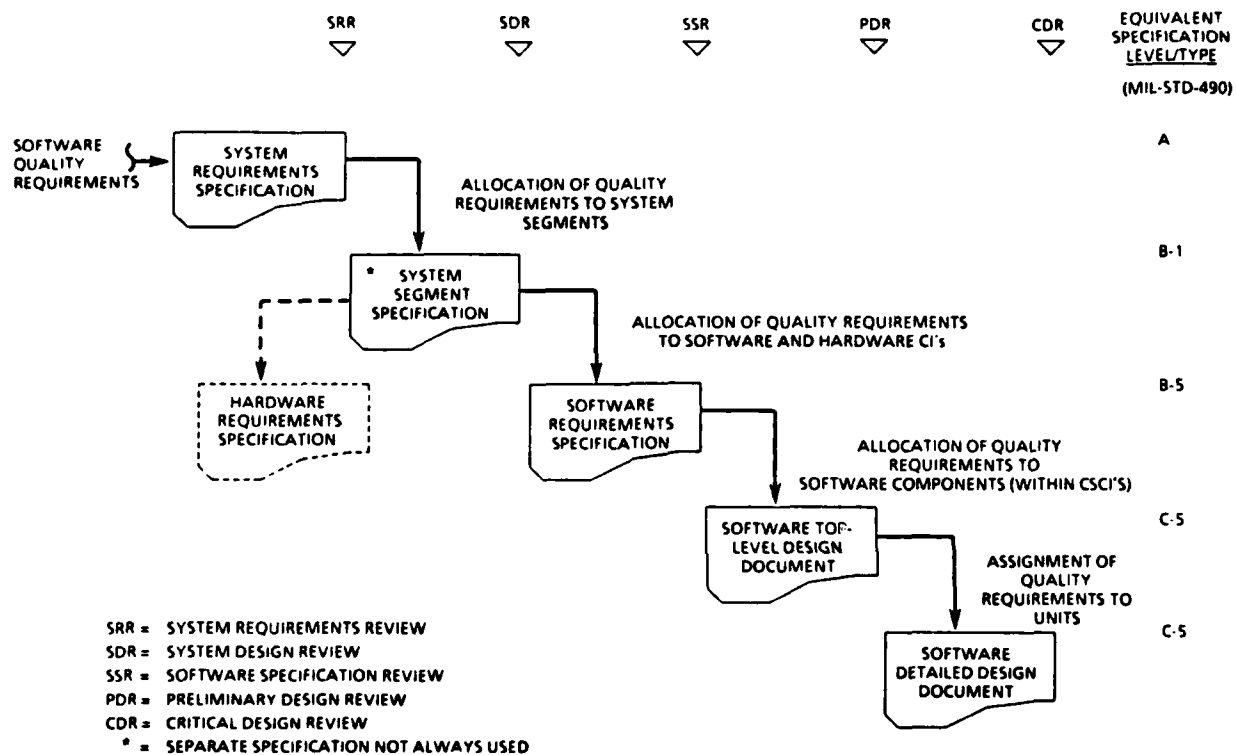


Figure 4.0-2 Flow of Software Quality Requirements

life cycle and achieved quality levels are evaluated periodically during development as was depicted in Figure 1.4-1. The methodology and guidebooks can also be used outside the life-cycle context to evaluate particular products such as a specification, design document, source code, or proposal. The purpose might be to evaluate reliability or maintainability of an operational product to determine if it is suitable for an application, to evaluate and compare quality levels of two products for purchasing, or to determine reusability of an operational product as an aid in determining adaptation costs for a new application. The purpose might also be to evaluate quality aspects of new-business proposals or system specifications to help determine a competitive contract award.

The methodology is similar regardless of context. Select important factors, criteria, and metrics. Select appropriate worksheets. Collect data and analyze results. Factor selection should be simplified for applications outside the life-cycle context because it is unlikely that factor cost trades would be performed; however, it is very important that factor interrelationships still be considered to avoid misinterpreting factor scores (explanation in Sec. 4.1.3). Criteria and metrics selections follow factor selection and should consider environmental and application particulars.

Selecting appropriate worksheets requires care to ensure desired results. In using the methodology for a new project with distinct development phases and reviews, a set of products is available at each review point. The metric worksheets are designed to be applied to these products. The products assumed to exist at the end of each software development phase are identified by title and data item description (DID) number in Table 3.4-2. To use the worksheets outside this life-cycle context, the product being evaluated should be matched as closely as possible to products identified in Table 3.4-2, and then the corresponding worksheets can be selected. For example, the technical portion of a proposal might correspond closest to a system and/or system segment specification or to a software requirements specification. Worksheet 0 and/or 1 would be chosen and appropriate questions selected. When the source code is available for an operational product, worksheets 4A and 4B would be used. If the detailed design documentation were available, worksheets 3A and 3B would also be used. Data collection and analysis results can be reported using the Software Quality Evaluation Report (see App. C).

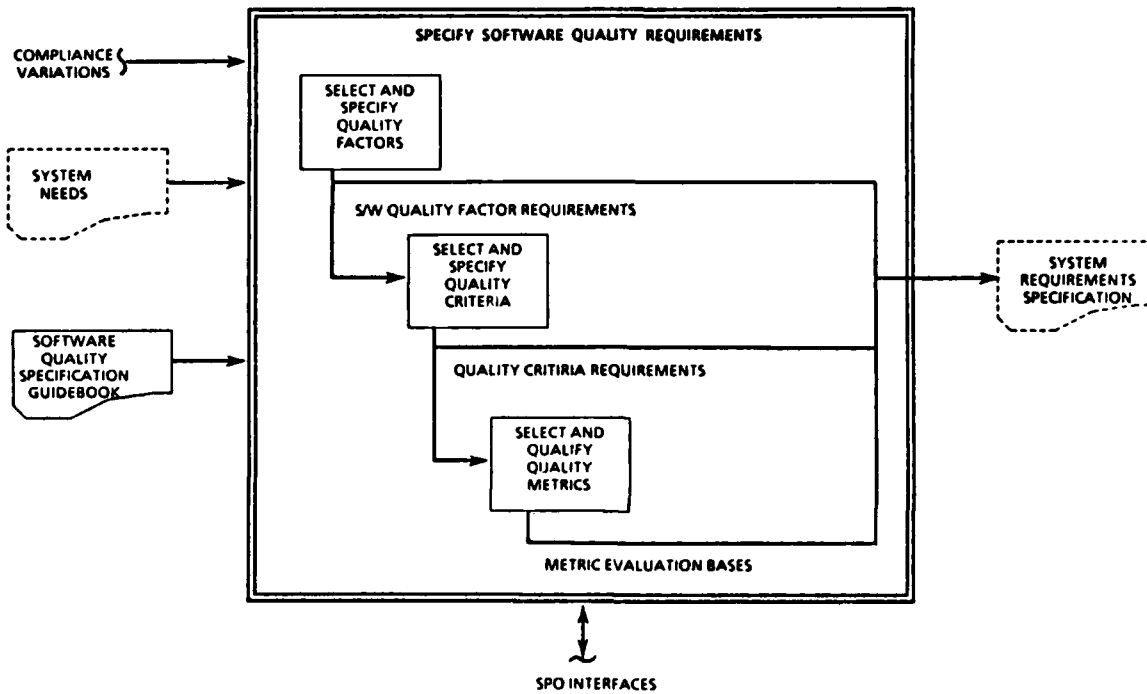


Figure 4.0-3 Procedures for Specifying Software Quality Requirements

Specification Procedural Overview. Software quality specification is divided into two separate processes (as is shown in Fig. 4.0-1): specify software quality requirements and assess compliance with requirements. The procedures for each of these are shown in Figures 4.0-3 and 4.0-4 and are arranged in chronological sequence—factors, criteria, and metrics—reflecting the three levels of detail of the hierarchical quality model.

Sections 4.1, 4.2, and 4.3 describe the detailed procedural steps for specifying software quality requirements. The steps are organized under three procedures (as is shown in Fig. 4.0-3): (1) select and specify quality factors, (2) select and specify quality criteria, and (3) select and qualify quality metrics. These procedures may all be performed at one time if enough details are known to be able to work with the criteria and metrics. Procedures may also be performed at different points in time but must be performed sequentially. That is, factors must be selected first, then criteria, then metrics. Also, it is recommended that the procedures be performed more than once; working through the procedures to a greater level of detail will often affect initial assumptions or decisions. The procedures only aid in a decision-making process for which there are no right answers, only answers best meeting user and system needs within cost and schedule constraints. Section 4.4 describes the procedural steps for assessing compliance with requirements (see Fig. 4.0-4).

4.1 SELECT AND SPECIFY QUALITY FACTORS

This procedure consists of four steps:

- a. Identify functions (step 1).
- b. Assign quality factors and goals (step 2).
- c. Consider interrelationships (step 3).
- d. Consider costs (step 4).

Steps 1 and 2 establish the quality goals; steps 3 and 4 consider the feasibility of achieving those goals. In the first step, each function which is supported by software and which will have separate quality requirements is identified. In step 2, quality factors are assigned to each separate function, and initial quality goals are established for each factor. In step 3, factor interrelationships (among the factors assigned to one function) are examined, and possible effects on quality goals are explored. In the fourth step,

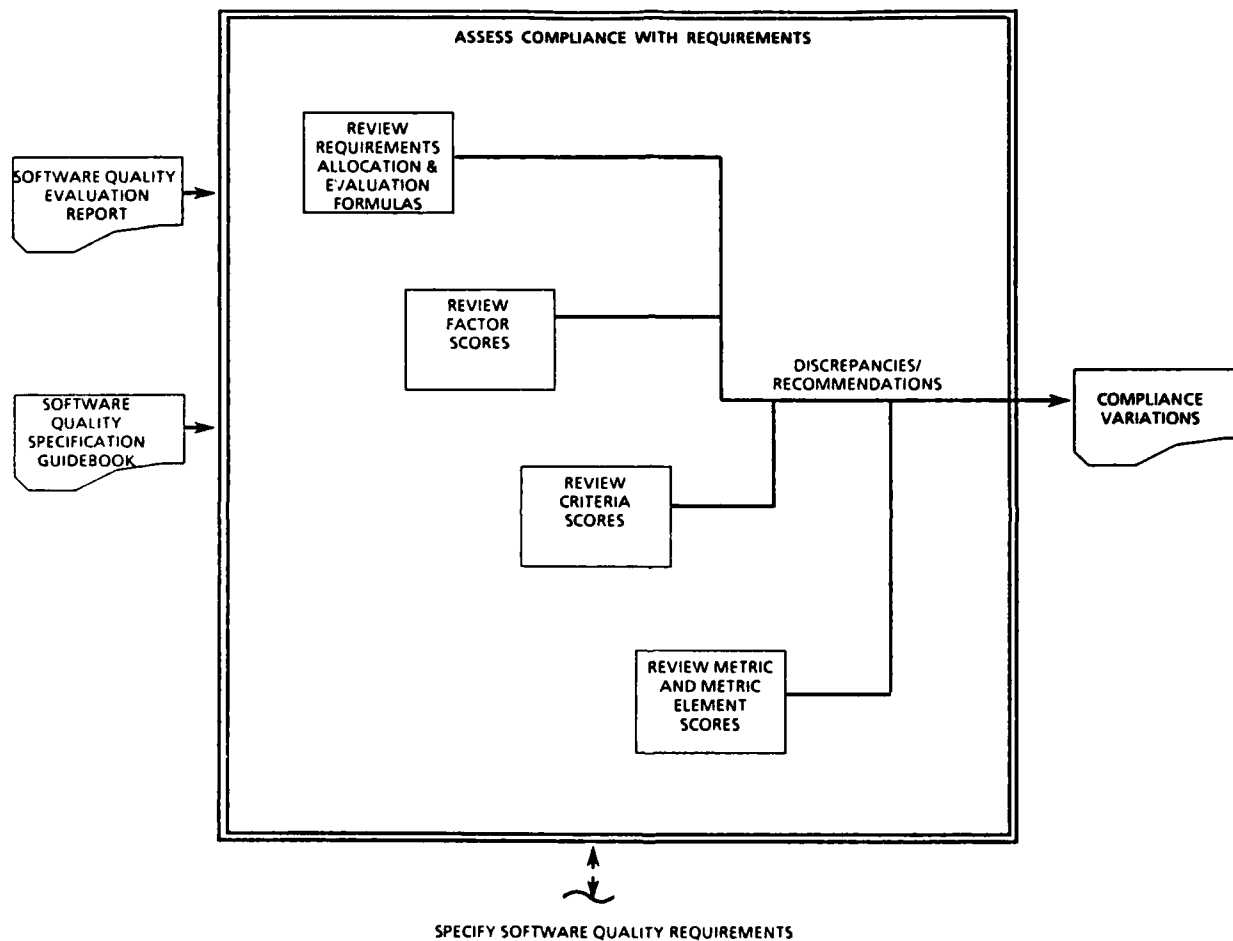


Figure 4.0-4 Procedures for Assessing Compliance with Requirements

factor life-cycle costs and cost variations due to interrelationships are examined, and possible effects on quality goals are considered.

4.1.1 Identify Functions (Step 1)

Step 1 of this procedure is to identify each function which is supported by software and which will have separate quality requirements.

Different system functions, and software supporting those functions, will likely have different quality needs. For example, a mission-dependent function should be more flexible (to accommodate a change in mission needs) than one that is not. The system description should be examined to identify all functions supported by software so that separate quality requirements can be specified for each. System-level functions are distinct operations performed by the system such as surveillance, identification, weapon assignment, communications, and guidance. Table 4.1.1-1 identifies an example command and control system and lists pertinent characteristics along with five system-level functions: surveillance and identification, threat evaluation, weapons assignment and control, battlestaff management, and communications. (This same system is used as the example throughout this procedure.)

All five functions are supported by software, and quality factor requirements for these functions can be allocated to operational software supporting them later in the life cycle as software becomes more well-defined. However, to obtain a complete set of quality requirements for the software, functions that are unique to software or that have separate quality needs should also be identified. By examining the system description, and with some knowledge of software development for command and control systems, these functions can be identified. The example system has four software-unique functions, as identified in Table 4.1.1-1: man-machine interface, executive, integrated test function, and mission training. Software supporting the man-machine interface is usually separated functionally for software development because it has unique performance and quality requirements associated with a human interface. Quality requirements should also be considered separately. Executive software interfaces operational software with computing hardware and hardware interfaces and is treated separately. Software supporting integrated (real-time) testing monitors system component operation and is treated separately. Mission training software is required to run in real time, using portions of the operational system, and is treated separately.

Table 4.1.1-1 Characteristics and Functions for Example System

SYSTEM:	Airborne Radar System
LIFE CYCLE:	15-20 years
COMPUTING SYSTEM:	Centralized, Redundant Processors

SYSTEM FUNCTIONS:

- Surveillance and Identification*
- Threat Evaluation
- Weapons Assignment/Control*
- Battlestaff Management
- Communications*

SOFTWARE - UNIQUE FUNCTIONS:

- Man-Machine Interface
- Executive*
- Integrated Test Function
- Mission Training

* = Function has external communications interface

4.1.2 Assign Quality Factors and Goals (Step 2)

In step 2, quality factors are assigned to each function supported by software (identified in step 1), and initial quality goals are established for each factor. Five areas are explored in accomplishing step 2: command and control quality concerns, system quality factors, quality requirements survey, complementary quality factors, and quality goals assignment.

4.1.2.1 Command and Control Quality Concerns

There are 13 quality factors, as defined in Section 3.0. Each function identified in the previous step can be assigned any one or more of these factors. Decisions should be made as to which qualities are needed for each function. System descriptions are likely to have vague software quality requirements. These procedures are written as though the system description has not addressed specific software quality requirements or has not addressed them comprehensively. Therefore, in examining the system description, judgment should be exercised in assigning the most appropriate factors to each function. The following information can be helpful.

Table 4.1.2.-1 lists typical command and control functions and software quality factors likely to be important for each function. These quality factors may or may not be important for a specific system and a specific application. Other quality factors may be important because of performance requirements or design constraints. For example, space, weight, or power constraints on the computing system may place emphasis on more efficient software. Other quality factors may be important because of basic characteristics of the application and or software environment. Table 4.1.2.-2 lists some application and environment characteristics and corresponding software quality factors likely to be important.

4.1.2.2 System Quality Factors

Some system descriptions contain requirements for system-level qualities such as availability, reliability, safety, transportability, and interchangeability. Those system-level quality requirements affecting system and software-unique functions identified in the previous step should be identified in terms of software quality factors.

Table 4.1.2-1 Important S/W Quality Factors for Major C2 Applications

<div> <div>S/W QUALITY FACTOR</div> <div>COMMAND AND CONTROL FUNCTION</div> </div>	EFFICIENCY	INTEGRITY	RELIABILITY	SURVIVABILITY	USABILITY	CORRECTNESS	MAINTAINABILITY	VERIFIABILITY	EXPANDABILITY	FLEXIBILITY	INTEROPERABILITY	PORTABILITY	REUSABILITY
INTELLIGENCE ELECTRONIC WARFARE (T) INTELLIGENCE DATA EXPLOITATION (T) TARGET ACQUISITION (T)		X	X X X	X			X	X	X	X	X		
SURVEILLANCE AND IDENTIFICATION DATA COLLECTION/REDUCTION (S) DYNAMIC GRAPHIC DISPLAY(S) TARGET RECOGNITION(S) THREAT DETECTION/IDENTIFICATION(S) THREAT DISPLAY (S) THREAT RESPONSE AIDS(S)	X X X X X X		X X X X X X	X						X	X		
ATTACK ASSESSMENT AND TACTICAL WARNING ATTACK & RESPONSE ASSESSMENT(S) ATTACK WARNING (T)	X X		X X										
DAMAGE ASSESSMENT DAMAGE DATA COLLECTION/REPORTING(S)	X		X							X	X		
SINGLE-INTEGRATED-OPERATIONS-PLAN SIOP OPTION SELECTION/EXECUTION(S) CONTROL AIDS(T) DECISION AIDS(T) DYNAMIC TARGETING/RETARGETING(S) TACTICAL PLANNING (T) WEAPON CONTROL/SELECTION(S)	X X	X X	X X X X	X	X	X			X	X X X X		X	X X
STRIKE ASSESSMENT STRIKE DATA COLLECTION/REPORTING(S)	X		X							X	X		
FORCE MANAGEMENT/RECONSTITUTION BATTLE FIELD MANAGEMENT(T) DYNAMIC FORCE MANAGEMENT (S)	X		X X				X		X	X X	X X	X	X X

NOTE (S) = STRATEGIC, (T) = TACTICAL

**Table 4.1.2-2 Examples of Application/Environment Characteristics and
Related Software Quality Factors**

APPLICATION/ENVIRONMENT CHARACTERISTICS	SOFTWARE QUALITY FACTORS
Human lives affected	Integrity Reliability Correctness Verifiability Survivability
Long life cycle	Maintainability Expandability
Experimental system or high rate of change	Flexibility
Experimental technology in hardware design	Portability
Many changes over life cycle	Flexibility Reusability Expandability
Real time application	Efficiency Reliability Correctness
On-board computer application	Efficiency Reliability Correctness Survivability
Processing of classified information	Integrity
Interrelated systems	Interoperability

Table 4.1.2-3 System/Software Quality Factor Correlation

ACQUISITION CONCERN/ QUALITY FACTOR SYSTEM QUALITY FACTOR	PERFORMANCE					DESIGN			ADAPTATION				
	EFFICIENCY	INTEGRITY	RELIABILITY	SURVIVABILITY	USABILITY	CORRECTNESS	MAINTAINABILITY	VERIFIABILITY	EXPANDABILITY	FLEXIBILITY	INTEROPERABILITY	PORTABILITY	REUSABILITY
AVAILABILITY			X	X			X						
EFFICIENCY	X				X								
INTEGRITY		X		—									
RELIABILITY		X	X			X							
SAFETY		X	X			X		X					
SURVIVABILITY		X		X							X		
TRANSPORTABILITY	X	X	X		X								
USABILITY			—		X								
CORRECTNESS						X		X					
MAINTAINABILITY				X			X						
VERIFIABILITY				X			X	X					
EXPANDABILITY	—							X	X	X			
FLEXIBILITY		—						X		X			
INTERCHANGEABILITY										X		X	X
INTEROPERABILITY											X	—	—
REUSABILITY								X		X			X

X = POSITIVE RELATIONSHIP
— = APPLICATION DEPENDENT

Table 4.1.2.-3 shows the correlation between 16 system quality factors and the 13 software quality factors. System quality factors are described in RADC-TR-83-175, volume I, Software Quality Measurement for Distributed Systems-Final Report. The following paragraphs discuss software quality factors likely to be important if a high quality rating has been specified for one of the system quality factors. (An example of critical function is one that would be selected for degraded mode operation.)

System availability is the portion of total operational time that the system performs or supports critical functions. High system availability implies high software reliability, survivability, and maintainability. High quality ratings for these factors ensure that the system will seldom fail, critical functions will continue to be performed in the event of a failure, and the fault will be quickly corrected.

System efficiency is the relative extent to which resources are utilized. High system efficiency implies both high software efficiency and high software usability. Software usability directly affects operator effectiveness and efficiency, and the system operator is a factor in measuring system efficiency.

System integrity is the extent to which the system will perform without failure due to unauthorized access to the system or system information. High system integrity implies high software integrity. In most applications, system integrity depends on the software and continued software functioning. In these applications software survivability would also impact system integrity.

System reliability is the extent to which the system will perform without any failure. High system reliability implies high software reliability, correctness, and integrity. Both software reliability and correctness contribute to the ability of the system to perform intended functions. High software integrity ensures that system reliability will not be adversely affected by accidental or deliberate unauthorized access to the software or data.

System safety is the extent to which the system will perform without causing damage or physical injury. High system safety implies high software correctness, reliability, integrity, and verifiability. High ratings for these factors ensure that the system will perform as specified, will seldom fail, and is secure from unauthorized access.

System survivability is the extent to which the system will perform and support critical functions without failure when a portion of the system is inoperable. High system survivability implies component and communication path redundancy and complex anomaly management. Complex anomaly management places emphasis on high software survivability. Emphasis is also placed on high software interoperability for redundancy and increased communications and, for networks with a variety of users, on high software integrity because of increased vulnerability to unauthorized access.

System transportability is the ease of effort for physically relocating the system. High system transportability implies low power, light weight, and compactness. These result in constraints on the computing system such as limited storage, emphasis on firmware rather than software, limited facilities for data entry and display, and wireless communication. These constraints place emphasis on software efficiency, integrity, and usability. Maintenance costs for the software of a transportable system are naturally high. This in turn places emphasis on software reliability to reduce probability of failure.

System usability is the relative effort for training or system operation. High system usability implies high software usability. In applications for which accuracy and precision affect the amount of time and effort to operate the system, the quality criteria accuracy would also be emphasized. (Accuracy is an attribute of the quality factor reliability.)

System correctness is the extent to which the system conforms to its specifications and standards. High system correctness implies both high software correctness and high software verifiability. The ability to verify software operation and performance against specifications and mission objectives aids in ensuring overall system correctness.

System maintainability is the ease of effort for locating and fixing a system failure. High system maintainability is enhanced by software that is easily maintainable and by software that aids in fault detection and isolation. This places emphasis on high software maintainability and on survivability to continue fault detection and isolation even when a portion of the system is inoperable.

System verifiability is the relative effort to verify the specified system operation and performance. High system verifiability implies component modularity, function modularity, fault isolation, high visibility of system operation through instrumentation and system displays, and diagnostic aids such as self-test capabilities. This places emphasis on high software verifiability and maintainability. Also, high software survivability would enable functions such as instrumentation, displays, and self-test to continue when a portion of the system is inoperable.

System expandability is the relative effort to increase system capability or performance by enhancing current functions or adding new functions. High system expandability implies component and function generality and modularity and implies spare system capacity. This emphasizes high software expandability, high software flexibility to incorporate enhancements and new functions, and high software verifiability to test changes. For a capacity-limited system, high software efficiency would be emphasized.

System flexibility is the ease of effort for changing system missions, functions, or data to satisfy other requirements. High system flexibility implies modular system components and generality of component functions. This requires flexible software that is modular and general and emphasizes change verification. In addition to high flexibility and verifiability, high integrity would also be emphasized when modifying functions, missions, or data could possibly compromise security.

System interchangeability is the relative effort to transform a system component for use in another environment. High system interchangeability implies that a family of systems (or subsystems) has components that are similar in function and that may be substituted for each other. This implies that there may be a need to reuse software from system to system (high reusability); to transfer software to another system configuration (high portability); or to modify software missions, functions, or data (high flexibility).

System interoperability is the relative effort to couple the system to another system. High system interoperability implies commonality of interface protocols, routines, and data representations. It also implies compatibility of interface equipment. This places emphasis on the high software interoperability and the ability to reuse software on interfacing systems (high reusability). For some applications it may be necessary to transfer software to an interfacing system (high portability).

Table 4.1.2-4 Software Quality Requirements Survey

INSTRUCTIONS:

The 13 software quality factors listed below represent aspects of software quality which are recognized as being important for certain software products. Use the attached Software Quality Factor Identification Form; list the system functions which are supported by software and the software-unique functions; and, for each function, indicate whether you consider each factor to be very important (goal of "E" for excellent quality), important (goal of "G" for good quality), somewhat important (goal of "A" for average quality), or either not important (blank) or not applicable (blank or "N/A").

ACQUISITION CONCERN	QUALITY FACTOR	DEFINITION
PERFORMANCE	EFFICIENCY	RELATIVE EXTENT TO WHICH A RESOURCE IS UTILIZED (I.E., STORAGE SPACE, PROCESSING TIME, COMMUNICATION TIME)
	INTEGRITY	EXTENT TO WHICH THE SOFTWARE WILL PERFORM WITHOUT FAILURES DUE TO UNAUTHORIZED ACCESS TO THE CODE OR DATA WITHIN A SPECIFIED TIME PERIOD
	RELIABILITY	EXTENT TO WHICH THE SOFTWARE WILL PERFORM WITHOUT ANY FAILURES WITHIN A SPECIFIED TIME PERIOD
	SURVIVABILITY	EXTENT TO WHICH THE SOFTWARE WILL PERFORM AND SUPPORT CRITICAL FUNCTIONS WITHOUT FAILURES WITHIN A SPECIFIED TIME PERIOD WHEN A PORTION OF THE SYSTEM IS INOPERABLE
	USABILITY	RELATIVE EFFORT FOR TRAINING OR SOFTWARE OPERATION (e.g., FAMILIARIZATION, INPUT PREPARATION, EXECUTION, OUTPUT INTERPRETATION)
DESIGN	CORRECTNESS	EXTENT TO WHICH THE SOFTWARE CONFORMS TO ITS SPECIFICATIONS AND STANDARDS
	MAINTAINABILITY	EASE OF EFFORT FOR LOCATING AND FIXING A SOFTWARE FAILURE WITHIN A SPECIFIED TIME PERIOD
	VERIFIABILITY	RELATIVE EFFORT TO VERIFY THE SPECIFIED SOFTWARE OPERATION AND PERFORMANCE
ADAPTATION	EXPANDABILITY	RELATIVE EFFORT TO INCREASE THE SOFTWARE CAPABILITY OR PERFORMANCE BY ENHANCING CURRENT FUNCTIONS OR BY ADDING NEW FUNCTIONS OR DATA
	FLEXIBILITY	EASE OF EFFORT FOR CHANGING THE SOFTWARE MISSIONS, FUNCTIONS, OR DATA TO SATISFY OTHER REQUIREMENTS
	INTEROPERABILITY	RELATIVE EFFORT TO COUPLE THE SOFTWARE OF ONE SYSTEM TO THE SOFTWARE OF ANOTHER SYSTEM
	PORTABILITY	RELATIVE EFFORT TO TRANSPORT THE SOFTWARE FOR USE IN ANOTHER ENVIRONMENT (HARDWARE CONFIGURATION AND/OR SOFTWARE SYSTEM ENVIRONMENT)
	REUSABILITY	RELATIVE EFFORT TO CONVERT A SOFTWARE COMPONENT FOR USE IN ANOTHER APPLICATION

System reusability is the relative effort to convert a system component for use in another application. High system reusability implies modularity of components and modularity and specificity of functions. This eases the tasks of selection and removal for reuse. It also implies that functions and components are general enough to be tailored to a new application. This places emphasis on high software reusability, high software flexibility to accommodate changes, and high software verifiability to test changes.

4.1.2.3 Quality Requirements Survey

System descriptions are often vague with respect to software quality requirements. Surveys can aid in defining specific software quality requirements. Tables 4.1.2-4 and 4.1.2-5 are example survey forms. Table 4.1.2-4 gives instructions for completing the survey and lists and defines the 13 software quality factors. Table 4.1.2-5 enables the respondent to identify each system or software-unique function supported by the software and to enter a quality factor level of importance—one for each of the 13 factors for each of the identified functions.

We recommend that the survey response be accompanied by a letter justifying each quality factor entry by referencing a system description paragraph or other requirements documentation (e.g., MIL-STD-1521 or DOD-STD-SDS) or by explaining the entry. Organizations responsible for software quality requirements and those potentially affected by low quality products should be surveyed. At a minimum, the Air Force using command and AFLC should be surveyed.

Entries in Table 4.1.2-5 represent survey results for the example system (see Tbl. 4.1.1-1). Survey respondents were familiar with the system description.

4.1.2.4 Complementary Quality Factors

This section discusses the effects of low quality levels among factors. Section 4.1.3 discusses the effects of high quality among factors.

Table 4.1.2-5 Software Quality Factor Identification Form - Survey Results

<div> <div>SOFTWARE QUALITY FACTOR</div> <div>SYSTEM OR SOFTWARE - UNIQUE FUNCTION</div> </div>	PERFORMANCE					DESIGN			ADAPTATION				
	EFFICIENCY	INTEGRITY	RELIABILITY	SURVIVABILITY	USABILITY	CORRECTNESS	MAINTAINABILITY	VERIFIABILITY	EXPANDABILITY	FLEXIBILITY	INTEROPERABILITY	PORTABILITY	REUSABILITY
				N/A								N/A	N/A
SURVEILLANCE AND IDENTIFICATION	E	E	E		E	E	E	G	G	G	G		
THREAT EVALUATION	G	A	E		E	E	E	G		G			
WEAPONS ASSIGNMENT /CONTROL	G	E	E		E	E	E	G	G	G	G		
BATTLESTAFF MANAGEMENT	A	A	A		E	E	G	G		G			
COMMUNICATIONS	E	E	E		E	E	E				G		
MAN-MACHINE INTERFACE		E	E		E	E	G		G	G			
EXECUTIVE	E	E	E				G				G		
INTEGRATED TEST FUNCTION			E			E	G		E				
MISSION TRAINING					E	E	G		E	G			

NOTE FOR GOAL ENTRIES.
 E = EXCELLENT
 G = GOOD
 A = AVERAGE
 BLANK or N/A = NOT IMPORTANT OR NOT APPLICABLE

Four quality factors are complementary to most other factors and should be considered when choosing important factors. These are reliability, correctness, maintainability, and verifiability, as shown in Table 4.1.2-6. Low quality levels for these factors can adversely effect measured (i.e., by factor rating formula) quality levels for other factors. Complementary means to fill out or to complete. For example, if a high quality level is specified for reliability, the specification is incomplete until a high quality level has also been specified for correctness and verifiability. The reasoning is that even if the metric scores were high for reliability, but the software was incorrect or difficult to verify, the actual reliability could be low (i.e., a high number of errors per lines of code) because of errors due to incorrect software or uncertainty in software verification.

Another example is that if a high quality level is specified for reusability, then high quality levels should also be specified for reliability, correctness, maintainability, and verifiability. The reasoning is that if metric scores were high for reusability, but the software was not highly reliable, incorrect, and/or difficult to maintain and verify, actual effort to reuse the software could be much higher than predicted by the reusability metric scores alone because of errors due to unreliable and incorrect software, difficulty in locating and fixing errors, and/or uncertainty in software verification.

Using factors (e.g., reliability) to view aspects of software quality enables a singular perspective on a particular quality concern; however this singular perspective is not always an independent perspective on quality, as is the case for complementary quality factors. Any project, regardless of the type of system or application, should include complementary factors in the quality requirements. Failure to do so could lead to a situation in which metric scores are not true indicators of the total quality present. Complementary factors should also be considered when attempting to validate metrics. Failure to do so could result in misinterpreting correlation results between metric scores and rating formula values.

4.1.2.5 Quality Goals Assignment

An initial set of quality goals should be assigned at this time. These goals should be based on considerations mentioned in previous sections:

Table 4.1.2-6 Complementary Software Quality Factors

COMPLEMENTARY QUALITY FACTOR QUALITY FACTOR SPECIFIED	EFFICIENCY	INTEGRITY	RELIABILITY	SURVIVABILITY	USABILITY	CORRECTNESS	MAINTAINABILITY	VERIFIABILITY	EXPANDABILITY	FLEXIBILITY	INTEROPERABILITY	PORTABILITY	REUSABILITY
EFFICIENCY													
INTEGRITY			*			*		*					
RELIABILITY						*		*					
SURVIVABILITY			*			*		*					
USABILITY			*			*	*	*					
CORRECTNESS													
MAINTAINABILITY			*			*		*					
VERIFIABILITY			*			*	*						
EXPANDABILITY			*			*	*	*					
FLEXIBILITY			*			*	*	*					
INTEROPERABILITY			*			*	*	*					
PORTABILITY			*			*	*	*					
REUSABILITY			*			*	*	*					

* = DEPENDENCY

Four quality factors are complementary to most other factors and should be considered when choosing important factors. These are reliability, correctness, maintainability, and verifiability, as shown in Table 4.1.2-6. Low quality levels for these factors can adversely effect measured (i.e., by factor rating formula) quality levels for other factors. Complementary means to fill out or to complete. For example, if a high quality level is specified for reliability, the specification is incomplete until a high quality level has also been specified for correctness and verifiability. The reasoning is that even if the metric scores were high for reliability, but the software was incorrect or difficult to verify, the actual reliability could be low (i.e., a high number of errors per lines of code) because of errors due to incorrect software or uncertainty in software verification.

Another example is that if a high quality level is specified for reusability, then high quality levels should also be specified for reliability, correctness, maintainability, and verifiability. The reasoning is that if metric scores were high for reusability, but the software was not highly reliable, incorrect, and/or difficult to maintain and verify, actual effort to reuse the software could be much higher than predicted by the reusability metric scores alone because of errors due to unreliable and incorrect software, difficulty in locating and fixing errors, and/or uncertainty in software verification.

Using factors (e.g., reliability) to view aspects of software quality enables a singular perspective on a particular quality concern; however this singular perspective is not always an independent perspective on quality, as is the case for complementary quality factors. Any project, regardless of the type of system or application, should include complementary factors in the quality requirements. Failure to do so could lead to a situation in which metric scores are not true indicators of the total quality present. Complementary factors should also be considered when attempting to validate metrics. Failure to do so could result in misinterpreting correlation results between metric scores and rating formula values.

4.1.2.5 Quality Goals Assignment

An initial set of quality goals should be assigned at this time. These goals should be based on considerations mentioned in previous sections:

Table 4.1.2-6 Complementary Software Quality Factors

COMPLEMENTARY QUALITY FACTOR QUALITY FACTOR SPECIFIED	EFFICIENCY	INTEGRITY	RELIABILITY	SURVIVABILITY	USABILITY	CORRECTNESS	MAINTAINABILITY	VERIFIABILITY	EXPANDABILITY	FLEXIBILITY	INTEROPERABILITY	PORTABILITY	REUSABILITY
EFFICIENCY													
INTEGRITY			*			*		*					
RELIABILITY						*		*					
SURVIVABILITY			*			*		*					
USABILITY			*			*	*	*					
CORRECTNESS													
MAINTAINABILITY			*			*							
VERIFIABILITY			*			*	*						
EXPANDABILITY			*			*	*	*					
FLEXIBILITY			*			*	*	*					
INTEROPERABILITY			*			*	*	*					
PORTABILITY			*			*	*	*					
REUSABILITY			*			*	*	*					

* = DEPENDENCY

- a. Command and control quality concerns.
- b. System quality factors.
- c. Quality requirements survey.
- d. Complementary quality factors.

Cost considerations and positive and negative factor interrelationships, which affect feasibility of the goals, will be considered in subsequent steps.

Table 4.1.2-7 represents the initial quality goals for the example airborne radar system. These goals are based on considerations a through d and include the complementary quality factors. The goal indicators used are E = excellent, G = good, A = average, and blank or N/A = not important or not applicable.

Numeric ranges for goal indicators will be discussed later. At this point in the specification process, an E goal would indicate a great deal of emphasis required for the quality factor, an expected high quality score, and correction of lower scoring areas of the software. A G goal would indicate emphasis on most aspects of the quality factor attributes with correction of only selected low scoring software areas. An A goal would indicate a desired awareness of the aspects of the quality factor and incorporation of quality considerations where convenient in the development methodology but with little or no cost impact. A blank or an N/A would indicate that this quality factor is either not important or not applicable.

The entries marked with an "*" in Table 4.1.2-7 are either new goal entries or entries changed to a higher goal as a result of considering complementary quality factors. (See Tbl. 4.1.2-5 for survey results.) An example is that the goals for verifiability were raised to E and new entries were made for several functions at the E level because of the E goals for reliability and usability.

4.1.3 Consider Interrelationships (Step 3)

In step 3, interrelationships among the factors assigned to a single system-level function are explored with respect to the technical feasibility of achieving the quality levels assigned as the initial set of quality goals. The initial set of quality goals was established in step 2. Step 3 and step 4 explore the feasibility of achieving those goals.

Table 4.1.2-7 Software Quality Factor Identification Form - Initial Goals

SYSTEM OR SOFTWARE UNIQUE FUNCTION	PERFORMANCE					DESIGN			ADAPTATION				
	EFFICIENCY	INTEGRITY	RELIABILITY	SURVIVABILITY	USABILITY	CORRECTNESS	MAINTAINABILITY	VERIFIABILITY	EXPANDABILITY	FLEXIBILITY	INTEROPERABILITY	CONTROLLABILITY	USABILITY
				N/A								N/A	N/A
SURVEILLANCE AND IDENTIFICATION	E	E	E		E	E	E	E*	G	G	G		
THREAT EVALUATION	G	A	E		E	E	E	E*		G			
WEAPONS ASSIGNMENT /CONTROL	G	E	E		E	E	E	E*	G	G	G		
BATTLESTAFF MANAGEMENT	A	A	G*		E	E	E*	E*		G			
COMMUNICATIONS	E	E	E		E	E	E	E*			G		
MAN-MACHINE INTERFACE		E	E		E	E	E*	E*	G	G			
EXECUTIVE	E	E	E			E*	G	E*			G		
INTEGRATED TEST FUNCTION			E			E	E*	E*	E				
MISSION TRAINING			E*		E	E	E*	E	E	G			

NOTE FOR GOAL ENTRIES
 E = EXCELLENT
 G = GOOD
 A = AVERAGE
 BLANK OR N/A =
 NOT IMPORTANT OR NOT
 APPLICABLE
 * = CHANGED

AD-A153 989

SPECIFICATION OF SOFTWARE QUALITY ATTRIBUTES VOLUME 2 2/2

SOFTWARE QUALITY SP. (U) BOOING AEROSPACE CO SEATTLE WA

T P BOWEN ET AL. FEB 85 D182-11678-2

UNCLASSIFIED

RADC-TR-85-37-VOL-2 F30602-82-C-0137

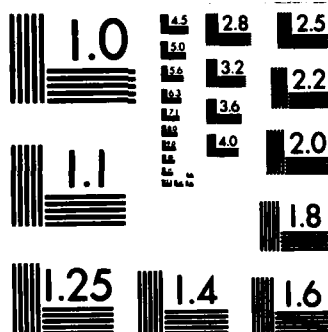
F/G 9/2

NL

END

FORMED

BY:



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Assigning more than one quality factor to a function (and therefore to software supporting that function) can, in some cases, have either a beneficial or an adverse effect, depending on the combination of factors that have been assigned. Some factors share common criteria; some have criteria that conflict with another factor; and some have criteria that benefit another factor. Four areas are explored in accomplishing step 3: shared criteria, beneficial and adverse relationships, quantification of relationships, and review of quality goals.

4.1.3.1 Shared Criteria

Shared criteria are those that are an attribute of more than one quality factor and can be identified using Table 3.2-1. For example, modularity is a criterion for eight of the 13 factors; generality is a criterion shared by three factors. In the example system, surveillance and identification has been assigned five quality factors sharing the criterion modularity (maintainability, verifiability, expandability, flexibility, and interoperability). The beneficial effect here is that modularity is built into the software only once, not five times; and metric data collection for modularity is performed only once. Therefore, costs associated with assigning factors that share common criteria are generally less than costs associated with assigning factors that do not share criteria. (Costs are considered in more detail in the next step.)

4.1.3.2 Beneficial and Adverse Relationships

Attribute criteria affecting another factor are identified in Table 4.1.3-1. Criteria that are basic attributes of a factor are identified with an x; criteria that are in a positive or cooperative relationship with a factor are identified with a triangle, and criteria that are in a negative or conflicting relationship with a factor are identified with an inverted triangle.

For example, operability is a criterion of usability and is shown as having a cooperative relationship with both maintainability and verifiability. The assertion is that the operability of usable software aids in software verification and maintenance, even though it is not an essential characteristic of verifiability and maintainability. The implication is that the desired rating for maintainability and verifiability (in terms of effort to fix or verify) will be easier to achieve if usability is also a specified quality factor.

Table 4.1.3-1 Effects of Criteria on Software Quality Factors

ACQUISITION CONCERN	ACQUISITION CONCERN			PERFORMANCE					DESIGN			ADAPTATION				
	FACTOR/ACRONYM			E F F I C I E N C Y	I N T E G R I T Y	R E L I A B I L I T Y	S U R V I V A B I L I T Y	U S A B I L I T Y	C O R R E C T N E S S	M A I N T A I N A B I L I T Y	V E R I F I A B I L I T Y	E X P A N D A B I L I T Y	F L E X I B I L I T Y	I N T E R O P E R A B I L I T Y	P O R T A B I L I T Y	R E U S A B I L I T Y
	CRITERION/ACRONYM			E F	I G	R L	S V	U S	C R	M A	V E	E X	F X	I P	P O	R U
P E R F O R M A N C E	ACCURACY	AC		▽												
	ANOMALY MANAGEMENT	AM		▽		X		△								
	AUTONOMY	AU					X									
	DISTRIBUTEDNESS	DN			▽		X									
	EFFECTIVENESS - COMMUNICATION	EC		X						▽	▽				▽	
	EFFECTIVENESS - PROCESSING	EP		X						▽	▽				▽	
	EFFECTIVENESS - STORAGE	ES		X						▽	▽				▽	
	OPERABILITY	OP						X		△	△					
	RECONFIGURABILITY	RE		▽			X			△		▽	▽		▽	
	SYSTEM ACCESSIBILITY	SS		▽	X											
	TRAINING	TN						X								
D E S I G N	COMPLETENESS	CP							X	△	△					△
	CONSISTENCY	CS							X	X	△	△	△			△
	TRACEABILITY	TC							X	X	△	△	△			△
	VISIBILITY	VS								X	X					
A D A P T A T I O N	APPLICATION INDEPENDENCE	AP													△	X
	AUGMENTABILITY	AT										X				
	COMMONALITY	CL		▽												
	DOCUMENT ACCESSIBILITY	DO			▽					△				X		X
	FUNCTIONAL OVERLAP	FO														
	FUNCTIONAL SCOPE	FS														X
	GENERALITY	GE		▽	▽	▽	▽					X	X	△	X	X
	INDEPENDENCE	ID		▽										X		X
	SYSTEM CLARITY	ST														X
G E N E R A L	SYSTEM COMPATIBILITY	SY			▽									X		
	VIRTUALITY	VR		▽								X				
	MODULARITY	MO		▽			X			X	X	X	X	X	X	X
	SELF-DESCRIPTIVENESS	SD		▽						X	X	X	X	X	X	X
	SIMPLICITY	SI		▽		X				X	X	X	X			X

NOTES. X = BASIC RELATIONSHIP
 △ = POSITIVE EFFECT
 ▽ = NEGATIVE EFFECT
 BLANK = NONE OR APPLICATION DEPENDENT

Anomaly management is an example of a criterion with a conflicting relationship. Anomaly management is a criterion of reliability and is shown as having a negative relationship with efficiency. The assertion is that the additional code required to perform anomaly management increases runtime and requires additional storage, thus decreasing potential efficiency. The implication is that the desired rating for efficiency (in terms of utilization of resources) will be more difficult (or more costly) to achieve if reliability is also a specified quality factor. Possible solutions to this type of conflict include:

- a. Spending the budget and schedule to try to achieve goals as high as possible for both factors.
- b. Lowering goals for one or the other factor.
- c. Allocating higher goals to the computing hardware (e.g., more efficient processor, more reliable processor) and possibly decreasing emphasis on high software quality goals (e.g., a more efficient processor to decrease emphasis on high software efficiency in achieving overall system efficiency and enabling a lower software quality goal for efficiency).

Another possibility is to decrease emphasis (goals) for the specific criterion that conflicts with another quality factor and increase emphasis on those that do not. This possibility will be explored in Section 4.2 in the discussion on criteria weighting.

4.1.3.3 Quantification Of Relationships

Table 4.1.3-2 details the rationale for each cooperative relationship identified in Table 4.1.3-1; shared criteria are identified as common for more than one factor. The degree of effect is noted as 1 = low, 2 = medium, and 3 = high. This will enable a quantification of the degree of effect caused by factor interrelationships. Table 4.1.3-3 details the rationale for negative factor interrelationships.

Table 2.2-2 summarizes, at the factor level, the positive and negative relationships in Table 4.1.3-1. Table 4.1.3-4 is in the same format as Table 2.2-2 and is an example quantification of interrelationships for factors assigned to the surveillance and identification function of the example system. For each factor assigned to this function, Tables 4.1.3-2 and 4.1.3-3 were consulted to identify the factors affected and the reason

Table 4.1.3-2 Positive Factor Interrelationships

NOTE: 1 = LOW
2 = MEDIUM
3 = HIGH
N/A = NOT APPLICABLE
AD = APPLICATION DEPENDENT

FACTOR SPECIFIED	FACTOR(S) AFFECTED	DEGREE AFFECTED	REASON
EFFICIENCY	NONE	N/A	N/A
INTEGRITY	AD	AD	AD (NOTE: HIGH INTEGRITY CAN REDUCE THE NUMBER OF ERRORS INCREASING THE RELIABILITY RATING)
RELIABILITY	SURVIVABILITY	1	⊙ THE CRITERION ANOMALY MANAGEMENT IS COMMON
	MAINTAINABILITY VERIFIABILITY EXPANDABILITY FLEXIBILITY REUSABILITY	1	⊙ THE CRITERION SIMPLICITY IS COMMON
	USABILITY	1	⊙ GOOD ANOMALY MANAGEMENT CAN REDUCE OPERATOR WORKLOAD
SURVIVABILITY	RELIABILITY	1	(SEE ⊙. ABOVE)
	MAINTAINABILITY VERIFIABILITY EXPANDABILITY FLEXIBILITY INTEROPERABILITY PORTABILITY REUSABILITY	1	⊙ THE CRITERION MODULARITY IS COMMON.
	USABILITY	1	(SEE ⊙. ABOVE)
	MAINTAINABILITY	1	THE RECONFIGURABILITY OF SURVIVABLE SOFTWARE CAN AID IN LOCATING PROBLEMS
USABILITY	MAINTAINABILITY VERIFIABILITY	1	THE COMMUNICATIVENESS OF USABLE SOFTWARE AIDS IN VERIFICATION AND MAINTENANCE OF THE SOFTWARE
CORRECTNESS	MAINTAINABILITY	1	⊙ THE CRITERION CONSISTENCY IS COMMON
	MAINTAINABILITY REUSABILITY	1	THE COMPLETENESS OF CORRECT SOFTWARE AIDS IN MAINTAINING AND REUSING THE SOFTWARE
	VERIFIABILITY EXPANDABILITY FLEXIBILITY REUSABILITY	1	⊙ THE CONSISTENCY OF CORRECT SOFTWARE AIDS IN VERIFYING THE SOFTWARE AND IN MODIFYING THE SOFTWARE FOR NEW USE.
	MAINTAINABILITY VERIFIABILITY EXPANDABILITY FLEXIBILITY REUSABILITY	1	THE TRACEABILITY OF CORRECT SOFTWARE AIDS IN VERIFICATION AND MAINTENANCE OF THE SOFTWARE AND IN MODIFYING THE SOFTWARE FOR NEW USE.

Table 4.1.3-2 Positive Factor Interrelationships (continued)

NOTE: 1 = LOW
2 = MEDIUM
3 = HIGH
N/A = NOT APPLICABLE
AD = APPLICATION DEPENDENT

FACTOR SPECIFIED	FACTOR(S) AFFECTED	DEGREE AFFECTED	REASON
MAINTAINABILITY	CORRECTNESS	1	(SEE ①, ABOVE)
	VERIFIABILITY	1	① THE CRITERION VISIBILITY IS COMMON
	SURVIVABILITY VERIFIABILITY EXPANDABILITY FLEXIBILITY INTEROPERABILITY PORTABILITY REUSABILITY	1	(SEE ①, ABOVE)
	VERIFIABILITY EXPANDABILITY FLEXIBILITY PORTABILITY REUSABILITY	1	① THE CRITERION SELF-DESCRIPTIVENESS IS COMMON
	RELIABILITY VERIFIABILITY EXPANDABILITY FLEXIBILITY REUSABILITY	1	(SEE ①, ABOVE)
	VERIFIABILITY EXPANDABILITY FLEXIBILITY REUSABILITY	1	(SEE ①, ABOVE)
VERIFIABILITY	MAINTAINABILITY	1	(SEE ①, ABOVE)
	SURVIVABILITY MAINTAINABILITY EXPANDABILITY FLEXIBILITY INTEROPERABILITY PORTABILITY REUSABILITY	1	(SEE ①, ABOVE)
	MAINTAINABILITY EXPANDABILITY FLEXIBILITY PORTABILITY REUSABILITY	1	(SEE ①, ABOVE)
	RELIABILITY MAINTAINABILITY EXPANDABILITY FLEXIBILITY REUSABILITY	1	(SEE ①, ABOVE)

Table 4.1.3-2 Positive Factor Interrelationships (continued)

NOTE: 1 = LOW
2 = MEDIUM
3 = HIGH
N/A = NOT APPLICABLE
AD = APPLICATION DEPENDENT

FACTOR SPECIFIED	FACTOR(S) AFFECTED	DEGREE AFFECTED	REASON
EXPANDABILITY	FLEXIBILITY REUSABILITY	1	⑨ THE CRITERION GENERALITY IS COMMON
	SURVIVABILITY MAINTAINABILITY VERIFIABILITY FLEXIBILITY INTEROPERABILITY PORTABILITY REUSABILITY	1	(SEE ⑩, ABOVE)
	MAINTAINABILITY VERIFIABILITY FLEXIBILITY PORTABILITY REUSABILITY	1	(SEE ⑩, ABOVE)
	RELIABILITY MAINTAINABILITY VERIFIABILITY FLEXIBILITY REUSABILITY	1	(SEE ⑩, ABOVE)
	INTEROPERABILITY	1	THE GENERALITY OF EXPANDABLE SOFTWARE AIDS IN INTERFACING WITH SOFTWARE OF OTHER SYSTEMS
FLEXIBILITY	EXPANDABILITY REUSABILITY	1	(SEE ⑩, ABOVE)
	SURVIVABILITY MAINTAINABILITY VERIFIABILITY EXPANDABILITY INTEROPERABILITY PORTABILITY REUSABILITY	1	(SEE ⑩, ABOVE)
	MAINTAINABILITY VERIFIABILITY EXPANDABILITY PORTABILITY REUSABILITY	1	(SEE ⑩, ABOVE)
	RELIABILITY MAINTAINABILITY VERIFIABILITY EXPANDABILITY REUSABILITY	1	(SEE ⑩, ABOVE)
	INTEROPERABILITY	1	THE GENERALITY OF FLEXIBLE SOFTWARE AIDS IN INTERFACING WITH SOFTWARE OF OTHER SYSTEMS

Table 4.1.3-2 Positive Factor Interrelationships (continued)

NOTE: 1 = LOW
2 = MEDIUM
3 = HIGH
N/A = NOT APPLICABLE
AD = APPLICATION DEPENDENT

FACTOR SPECIFIED	FACTOR(S) AFFECTED	DEGREE AFFECTED	REASON
INTEROPERABILITY	PORTABILITY REUSABILITY	1	⊕ THE CRITERION INDEPENDENCE IS COMMON
	SURVIVABILITY MAINTAINABILITY VERIFIABILITY FLEXIBILITY PORTABILITY REUSABILITY	1	(SEE ⊕. ABOVE)
PORTABILITY	INTEROPERABILITY REUSABILITY	1	(SEE ⊕. ABOVE)
	SURVIVABILITY MAINTAINABILITY VERIFIABILITY EXPANDABILITY INTEROPERABILITY REUSABILITY	1	(SEE ⊕. ABOVE)
	MAINTAINABILITY VERIFIABILITY EXPANDABILITY FLEXIBILITY REUSABILITY	1	(SEE ⊕. ABOVE)
REUSABILITY	EXPANDABILITY FLEXIBILITY	1	(SEE ⊕. ABOVE)
	INTEROPERABILITY PORTABILITY	1	(SEE ⊕. ABOVE)
	SURVIVABILITY MAINTAINABILITY VERIFIABILITY EXPANDABILITY FLEXIBILITY INTEROPERABILITY PORTABILITY	1	(SEE ⊕. ABOVE)
	MAINTAINABILITY VERIFIABILITY EXPANDABILITY FLEXIBILITY PORTABILITY	1	(SEE ⊕. ABOVE)
	RELIABILITY MAINTAINABILITY VERIFIABILITY EXPANDABILITY FLEXIBILITY	1	(SEE ⊕. ABOVE)
	PORTABILITY	1	THE DATA AND ARCHITECTURAL INDEPENDENCE OF REUSABLE SOFTWARE ENHANCES SOFTWARE PORTABILITY
	MAINTAINABILITY	1	WELL-STRUCTURED DOCUMENTATION WHICH IS EASY TO ACCESS IS AN AID IN MAINTAINING THE SOFTWARE
	INTEROPERABILITY	1	THE GENERALITY OF REUSABLE SOFTWARE AIDS IN INTERFACING WITH SOFTWARE OF OTHER SYSTEMS.

Table 4.1.3-3 Negative Factor Interrelationships

NOTE: 1 = LOW
2 = MEDIUM
3 = HIGH
N/A = NOT APPLICABLE

FACTOR SPECIFIED	FACTOR(S) AFFECTED	DEGREE AFFECTED	REASON
EFFICIENCY	MAINTAINABILITY	2	CODE WHICH IS OPTIMIZED FOR EFFICIENCY IS MORE DIFFICULT TO TEST AND TO MAINTAIN
	VERIFIABILITY	2	
	PORTABILITY	2	CODE WHICH IS OPTIMIZED FOR EFFICIENCY USUALLY DECREASES PORTABILITY
INTEGRITY	EFFICIENCY	2	THE ADDITIONAL RUN TIME AND STORAGE REQUIRED TO CONTROL ACCESS TO CODE AND/OR DATA DECREASES EFFICIENCY
RELIABILITY	EFFICIENCY	3	THE ADDITIONAL CODE REQUIRED TO PROVIDE ACCURACY AND TO PERFORM ANOMALY MANAGEMENT INCREASES RUN TIME AND REQUIRES ADDITIONAL STORAGE. THE USE OF AN HOL TO ACHIEVE SIMPLICITY CAN DECREASE EFFICIENCY.
SURVIVABILITY	EFFICIENCY	3	MODULAR, ANOMALY TOLERANT, RECONFIGURABLE SOFTWARE INCREASES RUN TIME AND REQUIRES ADDITIONAL STORAGE.
	INTEGRITY	2	THE DISTRIBUTEDNESS REQUIRED FOR SURVIVABLE SOFTWARE INCREASES THE RISK OF UNAUTHORIZED ACCESS
	FLEXIBILITY PORTABILITY REUSABILITY	1	THE RECONFIGURABILITY REQUIRED FOR SURVIVABLE SOFTWARE REDUCES ITS POTENTIAL EXPANDABILITY, FLEXIBILITY, PORTABILITY & REUSABILITY
USABILITY	EFFICIENCY	1	EASING AN OPERATOR'S TASK AND PROVIDING MORE USABLE OUTPUT REQUIRE MORE RUN TIME AND ADDITIONAL STORAGE.
CORRECTNESS	NONE	N/A	N/A
MAINTAINABILITY	EFFICIENCY	2	① MODULAR, SELF-DESCRIPTIVE, SIMPLE CODE RESULTS IN INCREASED OVERHEAD AND DECREASED OPERATING EFFICIENCY
VERIFIABILITY			
EXPANDABILITY	EFFICIENCY INTEGRITY RELIABILITY SURVIVABILITY	1	③ INCREASED GENERALITY OF THE CODE USUALLY RESULTS IN LESS EFFICIENCY, INCREASED VULNERABILITY TO UNAUTHORIZED ACCESS, A HIGHER NUMBER OF ERRORS, INCREASED DIFFICULTY IN PROVIDING ACCURACY, AND MORE COMPLEX ANOMALY MANAGEMENT
	EFFICIENCY	1	VIRTUAL STORAGE CAN INCREASE RUN TIME
	EFFICIENCY	2	(SEE ③ ABOVE)

Table 4.1.3-3 Negative Factor Interrelationships (continued)

NOTE: 1 = LOW
2 = MEDIUM
3 = HIGH
N/A = NOT APPLICABLE

FACTOR SPECIFIED	FACTOR(S) AFFECTED	DEGREE AFFECTED	REASON
FLEXIBILITY	EFFICIENCY INTEGRITY RELIABILITY SURVIVABILITY	1	(SEE ①, ABOVE)
	EFFICIENCY	1	(SEE ①, ABOVE)
INTEROPERABILITY	EFFICIENCY	2	THE USE OF STANDARD INTERFACE ROUTINES AND DATA REPRESENTATIONS INCREASES OVERHEAD AND REDUCES OPERATING EFFICIENCY.
	EFFICIENCY	1	① EMPHASIS ON SOFTWARE SYSTEM AND MACHINE INDEPENDENCE DECREASES POTENTIAL EFFICIENCY.
	INTEGRITY	1	COUPLED SYSTEMS HAVE MORE AVENUES OF ACCESS, MORE USERS, COMMON DATA REPRESENTATIONS, AND SHARED DATA AND CODE; THESE INCREASE THE RISK OF UNAUTHORIZED ACCESS
	EFFICIENCY	1	HIGHLY MODULAR CODE CAN REQUIRE MORE OVERHEAD, DECREASING OPERATING EFFICIENCY
PORTABILITY	EFFICIENCY	1	(SEE ①, ABOVE)
	EFFICIENCY	2	MODULAR, SELF-DESCRIPTIVE CODE RESULTS IN INCREASED OVERHEAD AND DECREASED OPERATING EFFICIENCY
REUSABILITY	INTEGRITY	3	WELL-STRUCTURED DOCUMENTATION WHICH IS EASILY ACCESSIBLE INCREASES THE RISK OF UNAUTHORIZED ACCESS
	EFFICIENCY INTEGRITY RELIABILITY SURVIVABILITY	1	(SEE ①, ABOVE)
	EFFICIENCY	1	(SEE ①, ABOVE)
	EFFICIENCY	2	(SEE ①, ABOVE)

Table 4.1.3-4 Factor Interrelationship Calculations

ACQUISITION CONCERN		PERFORMANCE					DESIGN			ADAPTATION				
ACQUISITION CONCERN	QUALITY FACTOR AFFECTED	EFFICIENCY	INTEGRITY	RELIABILITY	SURVIVABILITY	USABILITY	CORRECTNESS	MAINTAINABILITY	VERIFIABILITY	EXPANDABILITY	FLEXIBILITY	INTEROPERABILITY	PORTABILITY	REUSABILITY
	QUALITY FACTOR SPECIFIED													
					N/A								N/A	N/A
PERFORMANCE	EFFICIENCY	■						▽	▽				▽	
	INTEGRITY	▽ ₂	■											
	RELIABILITY	▽ ₃		■		△ ₁								
	SURVIVABILITY	N/A	▽		■	△		△		▽	▽		▽	▽
	USABILITY	▽ ₁				■		△ ₁	△ ₁					
DESIGN	CORRECTNESS						■	△ ₂	△ ₂	△ ₂	△ ₂			△
	MAINTAINABILITY	▽ ₂						■	△ ₁	△ ₁	△ ₁			△
	VERIFIABILITY	▽ ₂							■					
ADAPTATION	EXPANDABILITY	▽ ₄	▽ ₁	▽ ₁	▽					■		△ ₁		
	FLEXIBILITY	▽ ₂	▽ ₁	▽ ₁	▽						■	△ ₁		
	INTEROPERABILITY	▽ ₄	▽ ₁									■		
	PORTABILITY	N/A	▽										■	
	REUSABILITY	N/A	▽	▽	▽	▽		△				△	△	■
POSITIVE TOTALS △		N/A	N/A	N/A	N/A	+1	N/A	+3	+4	+3	+3	+2	—	—
NEGATIVE TOTALS ▽		-20	-3	-2	—	N/A	N/A	-2	-2	0	0	N/A	—	—

for and degree of effect. If the reason was valid for this system, the degree of effect was entered in the appropriate triangle. To remain conservative in estimating positive effects, only criteria relationships were considered. Effects of shared criteria were not quantified. Each column was totaled to indicate the degree of positive and negative interrelationship for each factor affected; the higher the score, the greater the effect on the factor. The cooperative effect is the highest for verifiability (+4), followed by maintainability, expandability, and flexibility (each with +3), then by interoperability (+2) and usability (+1). Major contributors to the cooperative effect are correctness (+8), maintainability (+3), and usability (+2).

The conflicting effect is, by far, the highest for efficiency (-20), followed by integrity (-3) and reliability, maintainability, and verifiability (each with -2). The score for flexibility is zero. All factors, except correctness, contribute to the conflicting effect. The major contributors are expandability (-6), interoperability (-5), and flexibility and efficiency (each with -4). The major conflict, as shown by the table, is needing software that would have low adaptation costs and also efficiently utilize resources in its intended application.

4.1.3.4 Review of Quality Goals

The final action in this step is to review initial quality goals established in step 2 (see Tbl. 4.1.2-7) in light of cooperative and conflicting relationships quantified in step 3 and to modify them if necessary. Two levels of goals were set for factors assigned to surveillance and identification: excellent and good. An excellent goal was set for seven factors: efficiency, integrity, reliability, usability, correctness, maintainability, and verifiability. A good goal was set for three factors: expandability, flexibility, and interoperability.

The main conflict in quality goals, as indicated by Table 4.1.3-4, is between the adaptation factors and efficiency. This conflict is important to note; however, it is not critical because goals for adaptation factors are good and not excellent, and the conflicting effect on efficiency is less. However, efficiency also conflicts with factors assigned an excellent goal: integrity, reliability, usability, maintainability, and verifiability; and the effect on efficiency is greater. This situation is critical and requires action because achieving the initial set of goals is not possible.

Table 4.1.3-5 Software Quality Factor Identification Form - Revised Goals

<div> <div>SOFTWARE QUALITY FACTOR</div> <div>SYSTEM OR SOFTWARE - UNIQUE FUNCTION</div> </div>	PERFORMANCE					DESIGN			ADAPTATION				
	EFFICIENCY	INTEGRITY	RELIABILITY	SURVIVABILITY	USABILITY	CORRECTNESS	MAINTAINABILITY	VERIFIABILITY	EXPANDABILITY	FLEXIBILITY	INTEROPERABILITY	PORTABILITY	REUSABILITY
SURVEILLANCE AND IDENTIFICATION	G*	E	E	N/A	E	E	E	E	G	G	G	N/A	N/A

NOTE FOR GOAL ENTRIES
 E = EXCELLENT
 G = GOOD
 A = AVERAGE
 BLANK OR N/A = NOT IMPORTANT OR NOT APPLICABLE
 * = CHANGED

The most obvious action, and the action taken for the example system, is to require very efficient processing hardware in order to alleviate the need for highly efficient software. This enables lowering the efficiency goal from excellent to good. Although this does not necessarily solve the problem, it changes the likelihood of achieving the goals from impossible to feasible.

These goals are established early in the system life cycle—at the same time that technical performance and design requirements are being established for the whole system. Thus, it is possible to increase efficiency requirements for processing hardware based on needs established by software quality goals. However, only a limited amount of efficiency is available from a given processor. Although the quality goal for efficiency has been lowered, efficiency still conflicts with most of the other factors. Several things might be done to ensure that overall goals can be met within budget constraints. For example, processing efficiency most often depends on a small percentage of software; high software efficiency goals could be assigned selectively to the software. Key units would be assigned high goals and other units assigned lower goals with the effect of satisfying overall efficiency requirements. Another possibility is to set quality goals at the criteria level in favor of nonconflicting criteria. For example, generality (a criterion of expandability) conflicts with three factors: efficiency, integrity, and reliability; whereas augmentability (also a criterion of expandability) does not conflict with any factor. A high quality goal can be assigned to augmentability and a lower goal set for generality so that average criteria scores will satisfy the factor goal. Please note that, although this technique will enable slightly higher scores for efficiency through minimizing conflict, it will reduce software expandability. Techniques for specifying criteria are explored in Section 4.2.

Table 4.1.3-5 shows the new set of quality goals for surveillance and identification. This set resulted from the analysis of this step of the procedure. Only the goal for efficiency was changed—from excellent to good—because of conflicts previously noted.

Table 4.1.4-1 Life-Cycle Quality Costs/Benefits

Life-cycle Activity/ Acquisition Phase	System/Software Requirements Analysis	Software Requirements Analysis through Detailed Design	Code and Unit Testing through Operational Testing and Evaluation	Production and Deployment	Product Adaptation
	Demonstration and Validation	Full Scale Development CDR		Production and Deployment	(Phases, as required)
<u>Potential Costs:</u>					
Specify Quality Requirements	X	X			{ Repeated, As Necessary }
Allocate Quality Requirements	X	X			
Design and Implement for Quality	(X)	X	X		
Evaluate Achieved Quality	X	X	X		
<u>Potential Cost Benefits:</u>					
Increased Quality Awareness	X	X	X	X	X
Better Quality Products	X	X	X		X
Early Problem Detection	(X)	X	X		X
Fewer Problems/Errors			X	X	X
Less Effort			(X)	X	X

Note: () = Project dependent

4.1.4 Consider Costs (Step 4)

In step 4, relative costs associated with specifying, designing, implementing, and measuring quality are explored. Relative costs with respect to a single quality factor are estimated for different parts of the life cycle, and the influence of factor interrelationships on relative costs are considered. The purpose of considering relative costs for factors is to enable a final decision on quality goals based on cost variations estimated for the factors chosen. The purpose is not to estimate life-cycle costs.

When using software QM technology in acquiring a product, additional costs are associated with specifying quality requirements, allocating those requirements to more detailed levels of requirements and design, designing and building quality into the product, and evaluating the quality level achieved for the product. There are also potential benefits when using QM technology. Benefits include an increased awareness of quality throughout the life cycle, higher quality products, early problem detection, fewer problems or errors, and less effort. Although difficult to quantify, these benefits are related to cost or cost avoidance. Costs and benefits will vary for different factors, different factor combinations, and different activities within the life cycle. Other benefits are also possible, including reduced risk. Costs and benefits for system acquisition phases are considered in the following paragraphs; details for each factor are considered later in this step.

4.1.4.1 Life-Cycle Quality Costs and Benefits

Table 4.1.4-1 summarizes potential quality costs and benefits for three system acquisition phases: demonstration and validation, full-scale development (FSD), and production and deployment. Because these potential costs and benefits are related to software, the software-related activities and phases are shown above the system acquisition phases. FSD is shown in two parts to further distinguish costs and benefits. A separate column is shown for adaptation of a product for a new use (e.g., expanded requirements, new application); acquisition phases are repeated as necessary.

The following paragraph describes the potential costs and benefits for software activities described in Table 4.1.4-1. Cost ranges for these activities are illustrated in Figure 4.1.4-1. Separate cost ranges are given for each factor. The cost ranges are from -3 to

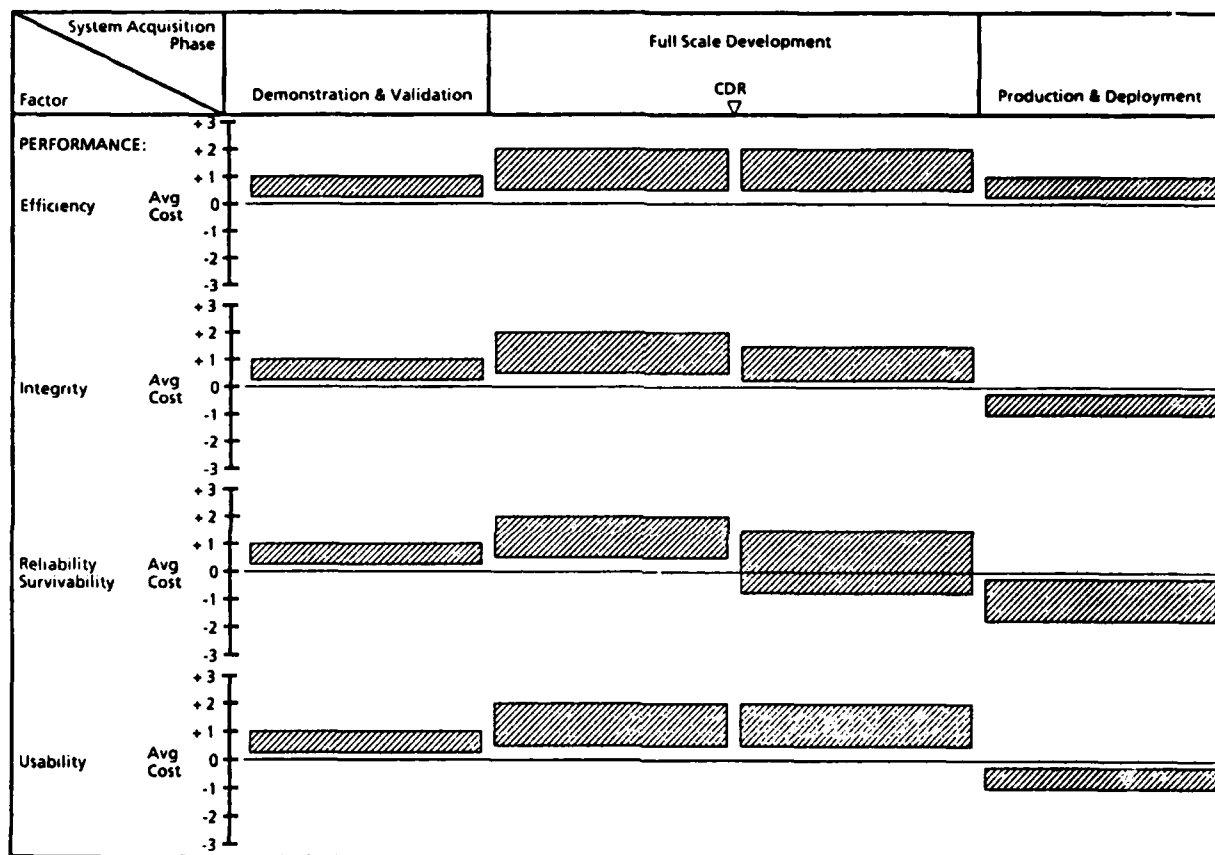


Figure 4.1.4-1 Quality Factor Life-Cycle Cost Ranges

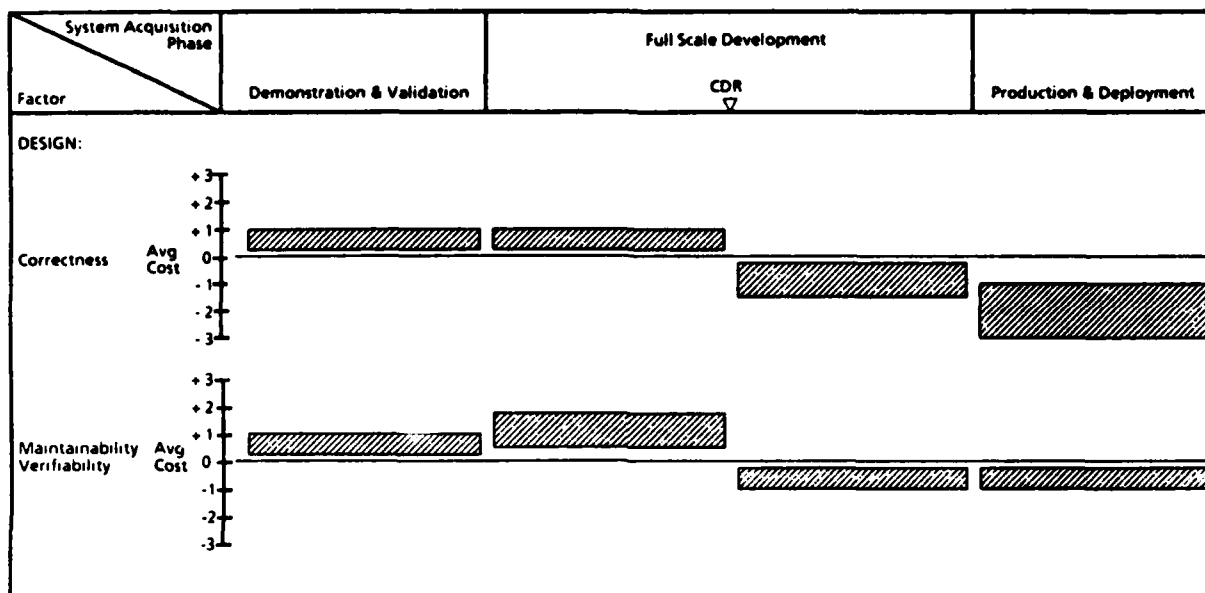


Figure 4.1.4-1 Quality Factor Life-Cycle Cost Ranges (continued)

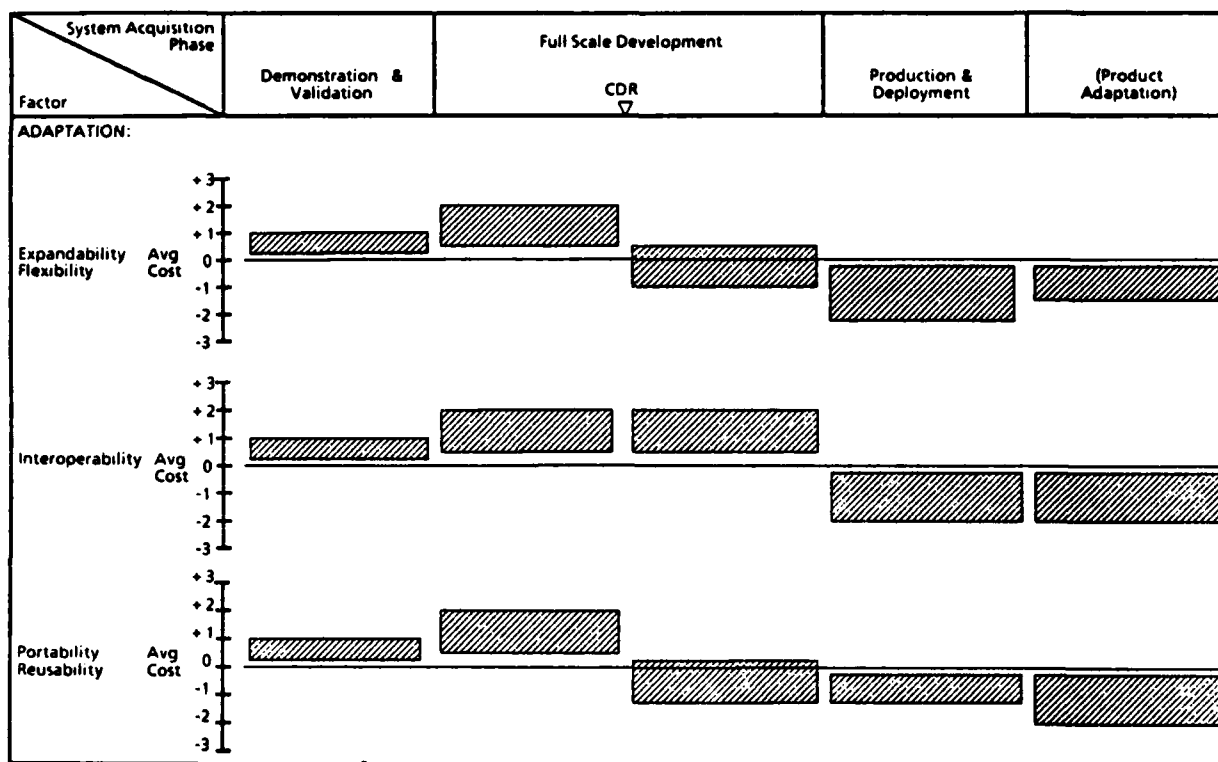


Figure 4.1.4-1 Quality Factor Life-Cycle Cost Ranges (continued)

+3 and are relative estimates based on additional quality activities that should be performed and on some benefits that are obvious for individual factors. Only software-related benefits are considered; other possible benefits (e.g., reduced risk, reduced loss of aircraft, and reduced system maintenance time.) are not.

Baseline cost (average cost) shown in Figure 4.1.4-1 reflects cost if the quality factor was not emphasized in the life cycle and only a nominal amount of quality was present. For example, the level of quality present resulted only from project development standards or from common practices of the development team. The cost ranges reflect the assumption that only one factor has been specified (i.e., no positive or negative interactions between factors). The effects of factor interactions with respect to cost are explored later in this section. To relate relative cost ranges in Figure 4.1.4-1 to costs for the project under consideration, estimate baseline cost for each acquisition phase and baseline cost variations due to quality factors by examining the quality considerations noted in the following paragraphs. A typical correlation would be that the cost range values correspond to percentages of cost for an acquisition phase. For example, specifying a high efficiency level would add up to 1% of the total cost of demonstration and validation for a command and control application. An example calculation for cost variations is given later in this section.

a. Demonstration and Validation

During demonstration and validation, software quality requirements are identified, specified, refined, and allocated to individual CSCIs. Early activities include specifying quality goals, coordinating with interfacing organizations in determining quality needs, and coordinating the consistency of quality requirements with technical performance and design requirements. As further knowledge of the system is gained, requirements, including software quality requirements, should be reviewed and refined; and quality requirements which have been levied against system-level functions and software-unique functions will be allocated to individual CSCIs performing those functions. As specifications are released, quality metrics are applied to the specifications, and the achieved quality level is assessed.

The cost range for all factors is from above zero to +1. Costs are likely to be on the low end for small, simple projects and on the high end for large, more complex projects for which quality requirements are more difficult to ascertain.

The potential benefit at this stage is an increased awareness of software quality considerations by using QM technology at the same time that technical requirements are being examined. These activities can complement each other and result in a comprehensive set of requirements and better quality software (and system) products (e.g., requirements specifications). If this phase includes designing and building a portion of the software and if quality requirements are also imposed on this software, there are additional costs. These costs are more typical of those incurred during FSD and are not reflected in the cost range estimates for this phase. An added benefit from this activity is early problem detection.

b. Full-Scale Development—Pre-CDR

The pre-CDR portion of FSD includes the latter portion of the software requirements analysis phase and the software design phases. During the latter portion of software requirements analysis, quality requirements are allocated to individual CSCIs and software functions and are checked for consistency with technical performance and design requirements. During the design phases, software quality requirements are allocated and assigned to successively lower design levels (i.e., top-level design and detailed design). Specified quality factors are emphasized in the design; and at each formal review when a software product is released, metrics are applied to the products, and achieved quality is assessed.

The cost range for all factors, except correctness, is from $+\frac{1}{2}$ to +2. Designing quality into software is the major portion of additional cost. For correctness, the cost range is from above zero to +1. For correctness, the cost range is less because quality is not built into software—design emphasis is on completeness, consistency, and traceability.

Potential benefits include continued increased awareness of quality considerations in software design, higher quality products at each design stage, and early problem detection by applying metrics to each product.

c. Full-Scale Development—Post-CDR

During the post-CDR portion of FSD, quality metrics are applied to incremental versions of software and to the final product. The cost range varies for different factors.

The cost range for efficiency is from $+\frac{1}{2}$ to +2. Efficient software tends to be more error prone and more costly to verify.

The cost range for integrity is from above zero to $+1\frac{1}{2}$. The additional code to perform access control and access auditing requires additional verification efforts.

The cost range for reliability and survivability is the same—from -1 (cost avoidance) to $+1\frac{1}{2}$. Additional verification effort is required because of increased emphasis on such things as accuracy, anomaly management, and reconfigurability. These additional costs can be offset by fewer errors and by automated software response to certain error conditions (because of emphasis on error handling and error avoidance during design).

The cost range for usability is from $+\frac{1}{2}$ to +2. Additional verification effort is required because of increased emphasis on operability and I/O communicativeness.

The cost range for correctness is from below zero to -1. Emphasizing correct software during design results in fewer problems related to incompleteness, misinterpretation of requirements, and inconsistencies within the design.

The cost range for maintainability and verifiability is from below zero to -1. Design emphasis on verifiable, maintainable software reduces testing complexity and simplifies locating and fixing errors.

The cost range for the adaptation factors expandability, flexibility, portability, and reusability is from $+\frac{1}{2}$ to -1. Some characteristics such as generality and augmentability add to the volume of code to be tested and increase the likelihood of errors. This cost is offset by characteristics such as modularity, self-descriptiveness, and simplicity, all of which simplify locating and fixing errors and aid in incorporating software changes during testing.

The cost range for interoperability is from $+\frac{1}{2}$ to +2. Additional effort is required to verify software commonality and compatibility with interfacing systems.

Potential benefits during the post-CDR portion of FSD are an increased awareness of quality considerations in software verification, higher quality software versions during testing (advantages for individual factors are noted in the above paragraphs), early problem detection by applying metrics, and fewer problems and errors because of increased emphasis on quality. Another possible benefit is that less effort is expended during this phase; this will depend on factor selection and on where actual costs fall within the cost ranges. Still another possible benefit is reduced risk because of greater emphasis on quality.

l. Production and Deployment

During production and deployment, the system is deployed into the field, and the software begins operation and maintenance. Software-related activities include operating the software, training personnel to use it, locating and fixing errors, and incorporating small changes. (Major changes, new applications, and new environments are considered under product adaptation.)

The cost range for efficiency is from above zero to +1. Highly efficient software tends to be more error prone (e.g., tightly written code often lacks modularity and tends to be more complex).

The cost range for integrity is from below zero to -1. Software with access limitations is likely to have fewer failures.

The cost range for reliability and survivability is from -½ to -2. Reliable, survivable software is likely to have fewer errors and failures.

The cost range for usability is from below zero to -1. Job time and duration of training are likely to be less for usable software.

The cost range for correctness is from -1 to -3. Software with a high degree of correctness is less likely to have problems related to incomplete designs, inconsistencies within the design, and misinterpretation of requirements. These problems can be expensive to correct after system deployment.

The cost range for maintainability and verifiability is from below zero to -1. Problems encountered after deployment are likely to be easier to locate and solutions easier to verify if the software is verifiable and maintainable. (Note that system maintainability usually includes the concept of improving the system by changing or expanding it. These capabilities are provided through software flexibility and expandability.)

The cost range for expandability and flexibility is from below zero to -2. Software characteristics such as generality, modularity, self-descriptiveness, and simplicity simplify locating and fixing errors, incorporating small changes into the software, and expanding basic software capabilities. Cost avoidance will vary depending on the quantity of expected changes and/or expansions.

The cost range for interoperability is from below zero to -2. Interfacing software with a system whose characteristics were known during development is simplified considerably. Cost avoidance will vary depending on the quantity of expected interfacing systems.

The cost range for portability and reusability is from below zero to -1. These factors are usually associated with major changes to the basic system or with redeployment of a new system that uses components from the original system; these are discussed under product adaptation. However, when most characteristics of these factors are present, the tasks of maintaining the software and incorporating small changes during production and deployment are simpler. Cost avoidance will vary depending on the expected quantity of changes.

e. Product Adaptation

Product adaptation is intended to address major changes to the software of an existing system for the same application, reuse of all or part of the software of an existing system in a new application, use of the software of an existing system in a new environment (i.e., hardware or operating system) for the same or a similar application, and converting the software of an existing system to interface with software of another system (when interface requirements are unknown during the development cycle). These efforts are all major changes to an existing system, and it is assumed that some or all of the development phases will be repeated before

Table 4.1.4-2 Cost Variations Calculation Form - Initial Estimate

LIFE-CYCLE ACTIVITY/ ACQUISITION PHASE/ % TOTAL COST QUALITY FACTOR/ GOAL		SYSTEM SOFTWARE REQUIREMENTS ANALYSIS	SOFTWARE REQUIREMENTS ANALYSIS THROUGH DETAILED DESIGN	CODE & UNIT TESTING THROUGH OPERATIONAL TESTING & EVALUATION	PRODUCTION AND DEPLOYMENT
		DEMONSTRATION AND VALIDATION	FULL SCALE DEVELOPMENT CDR		PRODUCTION AND DEPLOYMENT
		5 %	10%	15%	70%
EFFICIENCY	G	+ 1	+ 1½	+ 1½	+ ½
INTEGRITY	E	+ 1	+ 2	+ 1½	-1
RELIABILITY	E	+ 1	+ 2	+ ½	-2
SURVIVABILITY	N/A				
USABILITY	E	+ 1	+ 2	+ 2	-1
CORRECTNESS	E	+ 1	+ 1	- ½	-3
MAINTAINABILITY	E	+ 1	+ 2	- ½	-1
VERIFIABILITY	E	+ 1	+ 1½	- ½	- ½
EXPANDABILITY	G	+ 1	+ 1½	- ½	-1
FLEXIBILITY	G	+ 1	+ 1½	- ½	-1
INTEROPERABILITY	G	+ 1	+ 1½	+ 1½	-2
PORTABILITY	N/A				
REUSABILITY	N/A				
VARIATION TOTALS					

production and deployment. Cost ranges for the performance and design quality factors will be similar to those noted for FSD. Cost ranges for the adaptation quality factors are noted in the product adaptation column. The cost ranges show the potential for considerable savings when these qualities have been built into the software.

4.1.4.2 Cost Variation Estimates

At this point an initial estimate is made of cost variations for each factor, for each acquisition phase. This initial estimate should consider each factor separately (i.e., do not consider factor interactions). The effects of factor interactions will be considered later in this section.

Table 4.1.4-2 shows software cost variations estimated due to quality factors assigned to the surveillance and identification function of the example system. The purpose of estimating cost variations is to view the total influence of quality factors over the software life cycle. Estimates of software cost distribution over life-cycle phases should be made and percentages entered on the form. Quality goals, (refined in step 3, see Tbl. 4.1.3-5) should be entered next to the appropriate factor for easy reference. Next, enter a cost variation estimate for each factor in each phase using Figure 4.1.4-1 and rationale described in the preceding paragraphs as guidelines.

Cost distribution for surveillance and identification software in the example system follows the general cost trend for the total system: 5% of the total cost is in demonstration and validation, 25% is in FSD—10% pre-CDR and 15% post-CDR, and 70% is in production and deployment. Concept exploration costs were not considered; these costs are normally a very small percentage.

Cost distribution for the example system is typical of current command and control systems. Production and deployment costs usually range from 60% to 80% of the total. Post-CDR costs are usually higher than pre-CDR costs for FSD.

The following paragraphs describe the rationale for cost variation entries by phase in Table 4.1.4-2. These estimates are conservative.

a. Demonstration and Validation

The high end of the cost range, +1, was entered for each factor. This is a large, complex project, and costs for specifying the software quality requirements are likely to be independent of factor and goal level. In the example system, quality goal requirements were not levied on any prototype software; there were no additional quality-related costs.

b. Full-Scale Development—Pre-CDR

Costs during this period are both factor and goal dependent. The rationale for all entries was the same. If the goal is excellent, a value reflecting the high end of the cost range was entered for that factor. For example, the goal for reliability is excellent, and the top end of the cost range, +2, was entered. If the goal is good, a value near the middle of the cost range was entered for the factor. For example, the goal for efficiency is good and the middle of the cost range is $1\frac{1}{4}$; $1\frac{1}{2}$ was entered.

c. Full-Scale Development—Post-CDR

Costs during this period are also factor and goal dependent. For factors with goals of good, entries were made reflecting a value of near the middle of the cost range. For integrity and usability, with goals of excellent, entries were made reflecting a value at the top end of the cost range. For reliability, correctness, and maintainability, with goals of excellent, entries were made reflecting a value near the middle of the cost range because fewer errors were likely.

d. Production and Deployment

Efficiency is the only factor showing additional costs for this period; all other factors are below zero (cost avoidance). The entry for efficiency, with a goal of good, was $+\frac{1}{2}$. Entries for other factors with goals of good reflected a value near the middle of the (negative) cost range. Entries for factors with goals of excellent were at the extreme of the (negative) cost range because the full benefits of high quality levels were expected.

Costs for product adaptation can also be estimated at this time because these costs may influence the specified quality levels of certain factors.

4.1.4.3 Cost Effects of Factor Interrelationships

At this time, the effects of factor interrelationships on relative costs should be considered, and the initial cost variation estimates should be refined. Factor interrelationships have both positive and negative effects on cost just as they affect each other. Figure 4.1.4-2 shows the relative cost effects of positive factor interactions. The cost ranges shown are the same as those shown in Figure 4.1.4-1; arrows have been added for the appropriate phases to indicate the direction of the effect of interactions. Interrelationships among factors were summarized in Tables 2.2-2 and 4.1.3-1. The following paragraphs summarize the effects on relative costs by acquisition phase.

a. Demonstration and Validation

During this phase, many activities require only a little more time to perform for several factors as to perform for a single factor. For this reason, costs for an individual factor tend to decrease as the number of factors increases.

b. Full-Scale Development—Pre-CDR

During this period, the primary positive effect is shared criteria among factors. These characteristics are built into the software only once. The effect is reduced cost for specified factors sharing criteria. Three factors—efficiency, integrity, and usability—have no shared criteria. In the figure efficiency and integrity do not indicate a possibility of cost reduction. Usability is shown with a possible reduced cost range because of the positive effect of good anomaly management on reducing efforts for providing operable software. All other factors share criteria and are shown as having a possible cost-range reduction.

c. Full-Scale Development—Post-CDR

During this period, the primary positive effect is a reduced number of errors due to high reliability, correctness, and maintainability and due to errors automatically corrected by software with good anomaly management. All factors, except correctness, are shown with a possible reduced cost range; correctness is not affected by number of errors, and there is no additional effort associated with correctness after the design phases.

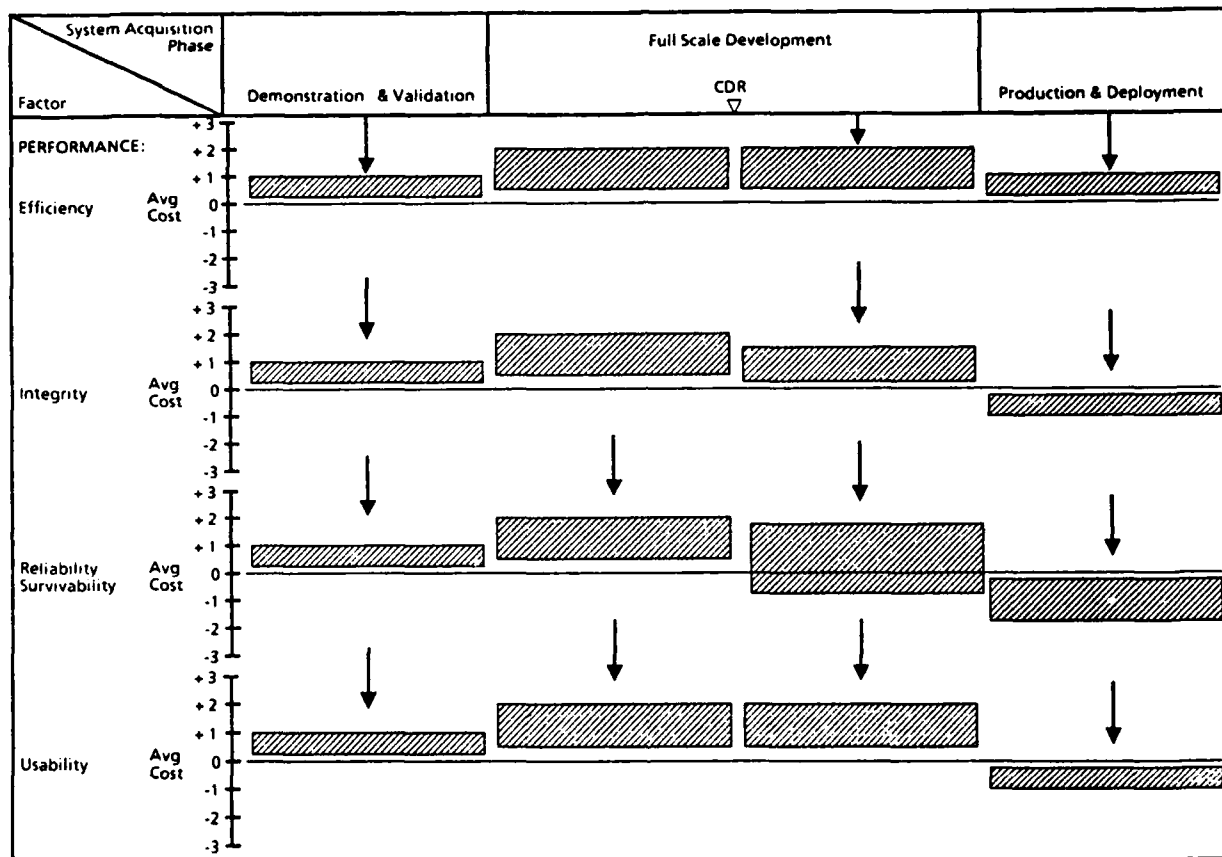


Figure 4.1.4-2 Cost Effects of Positive Factor Interrelationships

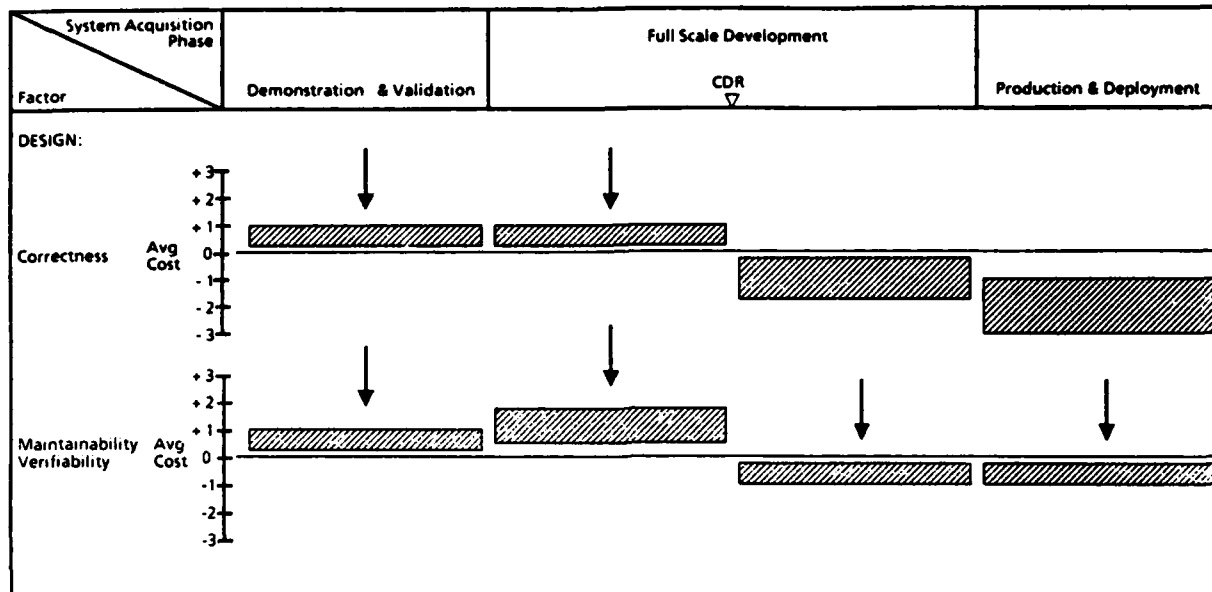


Figure 4.1.4-2 Cost Effects of Positive Factor Interrelationships (continued)

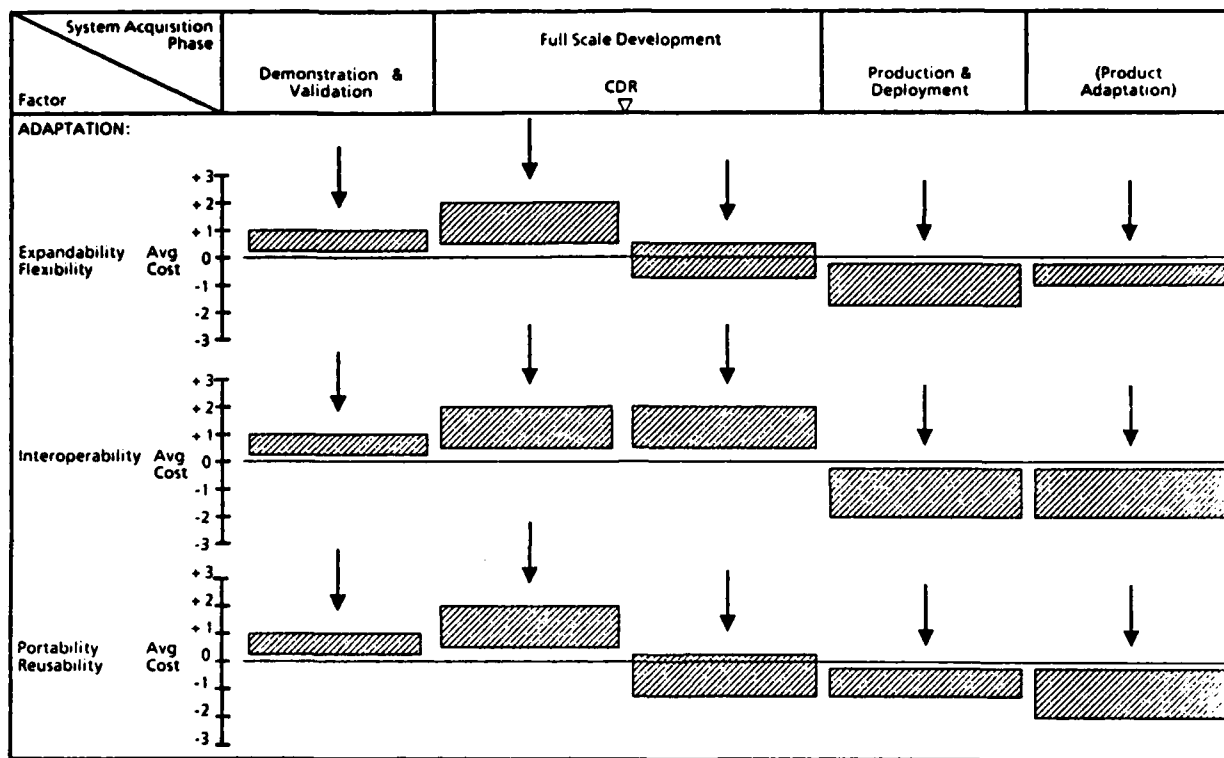


Figure 4.1.4-2 Cost Effects of Positive Factor Interrelationships (continued)

d. Production and Deployment

During this period, the primary positive effect is again a reduced number of errors. All factors, except correctness, are shown with a possible reduced cost range for the same reasons as noted in c.

e. Product Adaptation

Product adaptation is intended to address major changes to an existing system, and it is assumed that some or all development phases will be repeated before production and deployment. The positive effects on cost ranges for the performance and design quality factors are the same as those described for the development phases. All adaptation quality factors are shown with possible reduced cost ranges in the product adaptation column for the same reasons as noted above.

Figure 4.1.4-3 shows the cost effects of negative factor interrelationships. The cost ranges are the same as shown in Figure 4.1.4-1; arrows have been added to indicate the direction of the effect of interactions. The following paragraphs summarize negative effects on relative costs by acquisition phase.

a. Demonstration and Validation

No negative effects are shown for this phase because costs are predominantly independent of factor.

b. Full-Scale Development—Pre-CDR

During this period, only usability, correctness, and interoperability are shown as having no possible negative effect on the cost range because these are the only factors having no possible conflict with other factors. All other factors are shown as having a possible negative effect on cost range due to conflicts.

c. Full-Scale Development—Post-CDR

During this period, only usability, correctness, and interoperability are shown as having no possible negative effect on cost range.

d. Production and Deployment

During this period, only usability, correctness, and interoperability are shown as having no possible negative effect on cost range.

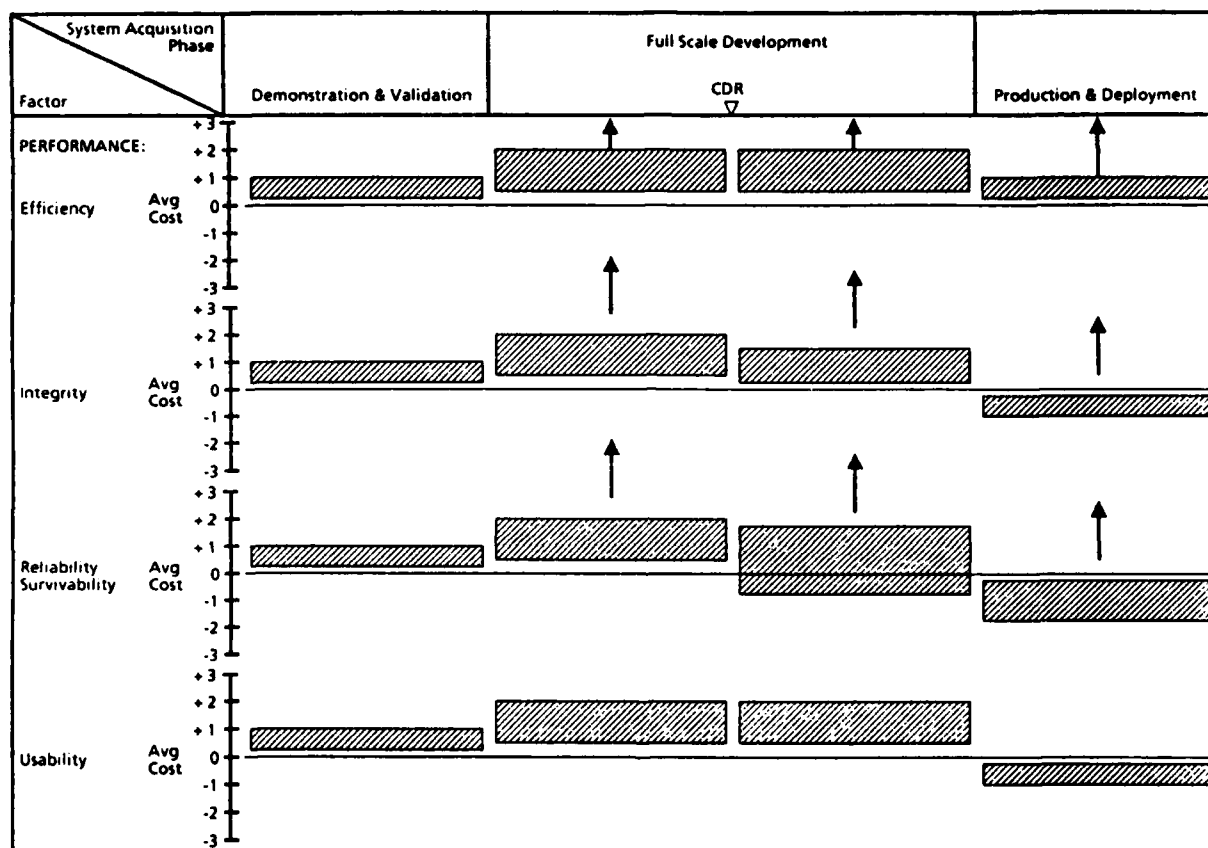


Figure 4.1.4-3 Cost Effects of Negative Factor Interrelationships

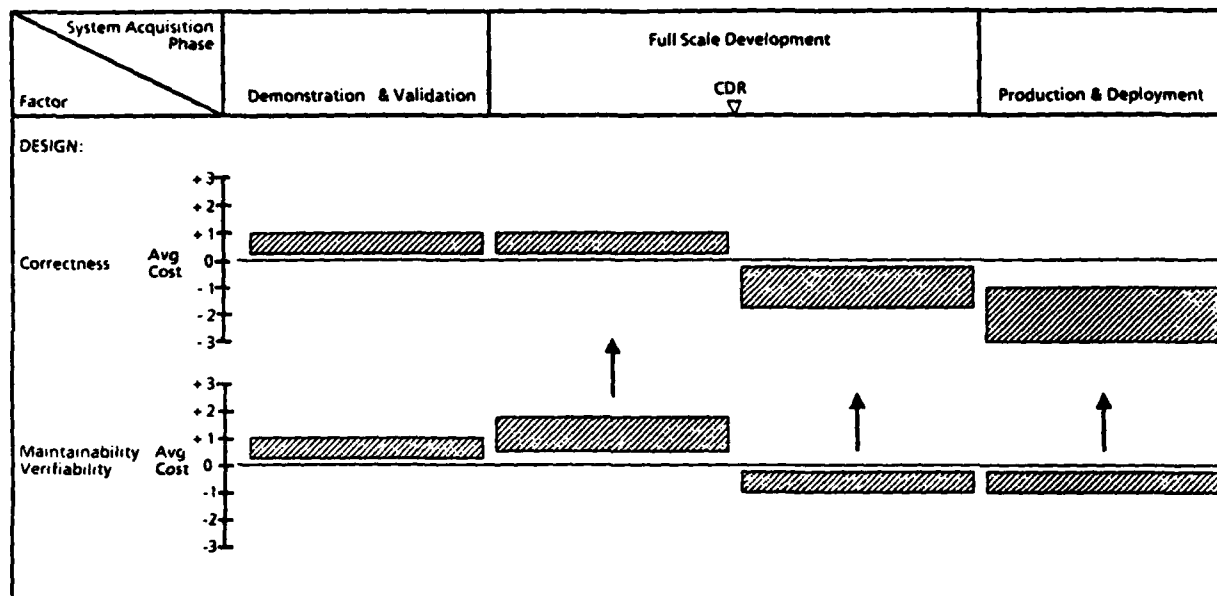


Figure 4.1.4-3 Cost Effects of Negative Factor Interrelationships (continued)

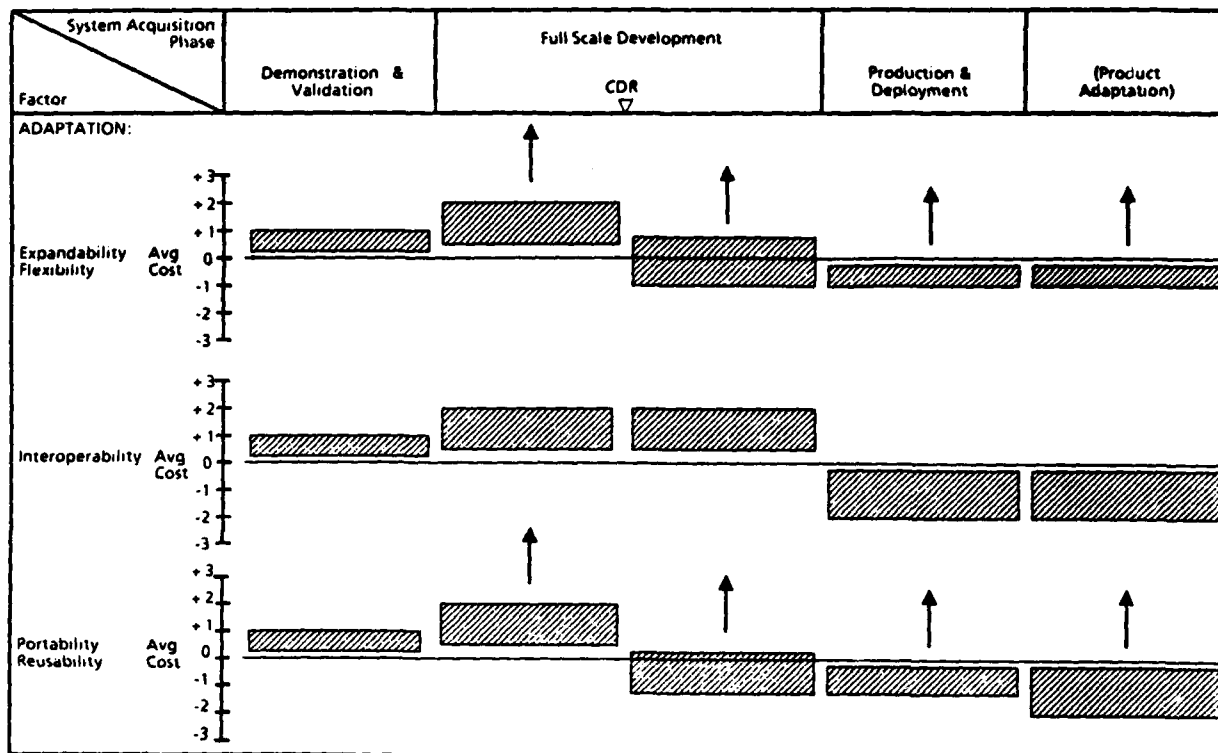


Figure 4.1.4-3 Cost Effects of Negative Factor Interrelationships (continued)

e. Product Adaptation

Product adaptation is intended to address major changes to an existing system, and it is assumed that some or all development phases will be repeated before production and deployment. The negative effects on cost range for the performance and design quality factors are the same as those effects for the development phases. Only interoperability is shown as having no possible negative effect on cost range in the product adaptation column.

Figure 4.1.4-3 considers only the effects of negative factor interrelationships. The information in this figure reflects the assumption that complementary quality factors (see sec. 4.1.2.4) have been specified where appropriate. If complementary factors have not been specified, further negative effects are possible.

Table 4.1.4-3 shows the cost variation estimates for surveillance and identification software of the example system. The estimates have been refined after considering positive and negative interactions of specified factors, summarized in Table 4.1.3-4. Rationale described in the preceding paragraphs was used for changing initial estimates. Changes can be noted by an asterisk. The following paragraphs summarize refined estimates by acquisition phase.

a. Demonstration and Validation

All entries were changed to ½ because of the quantity of factors.

b. Full-Scale Development—Pre-CDR

Efficiency was raised to +2, the top end of the cost range, because of conflicts with other factors. Usability, expandability, flexibility, and interoperability were lowered by ½ because of lack of conflict with other factors or shared criteria.

c. Full-Scale Development—Post-CDR

Efficiency was raised to +2 because the large number of conflicts with other factors would more than offset any cost reduction due to fewer errors. Usability and interoperability were reduced by ½ due to the effects of fewer errors and lack of conflicts.

Table 4.1.4-3 Cost Variations Calculation Form - Refined Estimate

LIFE-CYCLE ACTIVITY/ ACQUISITION PHASE/ % TOTAL COST QUALITY FACTOR/ GOAL		SYSTEM SOFTWARE REQUIREMENTS ANALYSIS	SOFTWARE REQUIREMENTS ANALYSIS THROUGH DETAILED DESIGN	CODE & UNIT TESTING THROUGH OPERATIONAL TESTING & EVALUATION	PRODUCTION AND DEPLOYMENT
		DEMONSTRATION AND VALIDATION	FULL SCALE DEVELOPMENT CDR ▽		PRODUCTION AND DEPLOYMENT
		5%	10%	15%	70%
EFFICIENCY	G	+ $\frac{1}{2}$ *	+2 *	+2 *	+1 *
INTEGRITY	E	+ $\frac{1}{2}$ *	+2	+1 $\frac{1}{2}$	-1
RELIABILITY	E	+ $\frac{1}{2}$ *	+2	+ $\frac{1}{2}$	-2
SURVIVABILITY	N/A				
USABILITY	E	+ $\frac{1}{2}$ *	+2	+1 $\frac{1}{2}$ *	-1
CORRECTNESS	E	+ $\frac{1}{2}$ *	+1	-\mathbf{\frac{1}{2}}	-3
MAINTAINABILITY	E	+ $\frac{1}{2}$ *	+2	-\mathbf{\frac{1}{2}}	-1
VERIFIABILITY	E	+ $\frac{1}{2}$ *	+1 $\frac{1}{2}$	-\mathbf{\frac{1}{2}}	-\mathbf{\frac{1}{2}}
EXPANDABILITY	G	+ $\frac{1}{2}$ *	+1 *	-\mathbf{\frac{1}{2}}	-1
FLEXIBILITY	G	+ $\frac{1}{2}$ *	+1 *	-\mathbf{\frac{1}{2}}	-1
INTEROPERABILITY	G	+ $\frac{1}{2}$ *	+1 *	+1 *	-2
PORTABILITY	N/A				
REUSABILITY	N/A				
VARIATION TOTALS		+5	+15 $\frac{1}{2}$	+4	-11 $\frac{1}{2}$

* = CHANGED

d. **Production and Deployment**

Efficiency was raised by $\frac{1}{2}$ due to the large number of conflicts with other factors.

The variation totals are added for each column in Table 4.1.4-3. Potential costs are highest for the FSD pre-CDR period; potential cost savings (avoidance) are shown for production and deployment. If variation totals are assumed to represent percentages of costs for that phase, an overall cost savings of approximately 5% is projected for the total life cycle $((0.05 \times 1.05) + (0.10 \times 1.155) + (0.15 \times 1.04) + (0.70 \times 0.885) = 0.0525 + 0.1155 + 0.156 + 0.6195 = 0.9435)$. This projected savings considers only software-related costs and benefits; system-related cost avoidance has not been projected.

Potential cost savings for product adaptation for a new use were not considered here. Although these types of costs and cost savings are outside the system life cycle, they may influence decisions on quality level for some factors and should be considered if possible.

4.1.4.4 Review of Quality Goals

At this time quality goals should be reviewed in light of life-cycle cost considerations of this step and revised as necessary. Table 4.1.4-4 lists the final goals for the surveillance and identification function of the example system. None of the goals were changed (from Tbl. 4.1.3-5); however, cost limits were placed on the efforts for providing six of the factors: efficiency, integrity, reliability, usability, maintainability, and verifiability. These cost limits are primarily intended to limit the level of effort during design phases when factor conflicts show the greatest effect.

Quality factor requirements should be specified quantitatively, as a value or value range, and should reflect system and user needs and budget constraints. No industry standards have been established, and judgement should be exercised when assigning values. Values should not be unrealistically high because this situation can drain resources. However, during the software development cycle, it is easier for the development contractor to respond to lowering the goals than to raising the goals. We recommend using value ranges and believe that the following ranges are realistic for a typical command and control application: E—from 0.90 to 1, G—from 0.80 to 0.89, and A—from 0.70 to 0.79. Higher ranges for selected factors may be appropriate in acquisitions involving space applications or nuclear armaments.

Table 4.1.4-4 Software Quality Factor Identification Form - Final Goals

<div> <div>SOFTWARE QUALITY FACTOR</div> <div>SYSTEM OR SOFTWARE- UNIQUE FUNCTION</div> </div>	PERFORMANCE					DESIGN			ADAPTATION				
	EFFICIENCY	INTEGRITY	RELIABILITY	SURVIVABILITY	USABILITY	CORRECTNESS	MAINTAINABILITY	VERIFIABILITY	EXPANDABILITY	FLEXIBILITY	INTEROPERABILITY	PORTABILITY	REUSABILITY
SURVEILLANCE AND IDENTIFICATION	G*	E*	E*	N/A	E*	E	E*	E	G	G	G	N/A	N/A

NOTE FOR GOAL ENTRIES
 E = EXCELLENT
 G = GOOD
 A = AVERAGE
 BLANK OR N/A =
 NOT IMPORTANT OR NOT
 APPLICABLE
 * = COST LIMITS REQUIRED

4.2 SELECT AND SPECIFY QUALITY CRITERIA

Select and specify quality criteria is the second of three procedures for identifying software quality requirements (see Fig. 4.0-3). This procedure consists of three steps:

- a. Select criteria.
- b. Assign weighting formulas.
- c. Consider interrelationships.

Steps 1 and 2 establish the relationships of criteria to factors. Step 3 considers the feasibility of achieving quality factor goals established in Section 4.1 using these criteria relationships.

4.2.1 Select Criteria (Step 1)

Step 1 of this procedure is to identify all criteria which are attributes of each factor for which final goals were established in the previous procedure (see Sec. 4.1). Final goals were established for 10 of the 13 quality factors for the example surveillance and identification function (see Fig. 4.1.4-4). Use Table 3.2-1 to identify all criteria which are attributes of each applicable factor. For example, there are three criteria for reliability—accuracy, anomaly management, and simplicity.

4.2.2 Assign Weighting Formulas (Step 2)

Step 2 of this procedure is to assign a weighting formula for each applicable quality factor. A weighting formula shows a specific relationship between the factor and its attribute criteria; each criterion is assigned a weighting value to indicate its percentage contribution to the overall factor goal. Table 4.2-1 lists the weighting formulas for factors of the surveillance and identification example. The formula for reliability indicates the percentage contribution of three criteria: accuracy—40%, anomaly management—30%, and simplicity—30%.

Weighting formulas are significant when scoring and when specifying requirements. When scoring, a criterion score is multiplied by the weighting value in calculating a factor score. A higher weighting value places more emphasis on a high criterion score in achieving the factor scoring goal. A lower weighting value places less emphasis on

Table 4.2-1 Criteria Weighting Formula Form - Initial Weighting

Factor	Weighting Formula
(EFFICIENCY)	$= 0.1 (EC) + 0.8 (EP) + 0.1 (ES)$
(INTEGRITY)	$= 1.0 (SS)$
(RELIABILITY)	$= 0.4 (AC) + 0.3 (AM) + 0.3 (SI)$
(SURVIVABILITY) N/A	$= (AM) + (AU) + (DI) + (RE) + (MO)$
(USABILITY)	$= 0.5 (OP) + 0.5 (TN)$
(CORRECTNESS)	$= 0.4 (CP) + 0.3 (CS) + 0.3 (TC)$
(MAINTAINABILITY)	$= 0.2 (CS) + 0.2 (VS) + 0.2 (MO) + 0.2 (SD) + 0.2 (SI)$
(VERIFIABILITY)	$= 0.25 (VS) + 0.25 (MO) + 0.25 (SD) + 0.25 (SI)$
(EXPANDABILITY)	$= 0.2 (AT) + 0.2 (GE) + 0 (VR) + 0.2 (MO) + 0.2 (SD) + 0.2 (SI)$
(FLEXIBILITY)	$= 0.25 (GE) + 0.25 (MO) + 0.25 (SD) + 0.25 (SI)$
(INTEROPERABILITY)	$= 0.2 (CL) + 0.2 (FO) + 0.2 (ID) + 0.2 (SY) + 0.2 (MO)$
(PORTABILITY) N/A	$= (ID) + (MO) + (SD)$
(REUSABILITY) N/A	$= (AP) + (DO) + (FS) + (GE) + (ID) + (ST) + (MO) + (SD) + (SI)$

NOTE: () = Score

achieving a high criterion score. When specifying requirements, weighting values indicate the amount of emphasis on a software characteristic (criterion) in developing the product.

System goals and requirements should be reviewed when assigning weighting values, and any significant system needs should be reflected in weighting formulas. Weighting formulas in Table 4.2-1 show approximately equal weighting for most criteria. There are two exceptions. Virtuality (an attribute of expandability) is weighted zero; this criterion is primarily applicable to networks and was not considered appropriate for the example system. The criteria of efficiency are not weighted equally; processing effectiveness is weighted much higher than communication and storage effectiveness. For the surveillance and identification function, processing speed is far more critical than communication speed and storage space.

4.2.3 Consider Interrelationships (Step 3)

Step 3 of this procedure considers the effect of positive and negative relationships between criteria and factors on the feasibility of achieving quality factor goals. A negative relationship between a criterion and factor may affect the feasibility of achieving a high enough criterion score to satisfy the factor scoring goal which reflects the system needs. Adjusting the weighting formulas communicates to the developer desired areas of emphasis and areas where compromises can be made in software characteristics for the sake of achieving the overall factor goal.

Table 4.2-2 shows revised weighting formulas for the example surveillance and identification function. Formulas have been revised for four factors: maintainability, verifiability, expandability, and flexibility. Changes were based on the interrelationships shown in Table 4.1.3-1 and on rationale similar to that described in Table 4.1.3-3. Generality is in conflict with three specified factors: efficiency, integrity, and reliability. The weighting values for generality were lowered to 0.1 in the formulas for expandability and flexibility. The message to the developer is that, although generality is to be considered, lower scores will be tolerated because of significant conflicts.

Because weighting values for generality were lowered, weighting values for other criteria must be increased. For expandability, the weighting value for augmentability

Table 4.2-2 Criteria Weighting Formula Form - Revised Weighting

Factor	Weighting Formula
(EFFICIENCY)	= 0.1 (EC) + 0.8 (EP) + 0.1 (ES)
(INTEGRITY)	= 1.0 (SS)
(RELIABILITY)	= 0.4 (AC) + 0.3 (AM) + 0.3 (SI)
(SURVIVABILITY) N/A	= (AM) + (AU) + (DI) + (RE) + (MO)
(USABILITY)	= 0.5 (OP) + 0.5 (TN)
(CORRECTNESS)	= 0.4 (CP) + 0.3 (CS) + 0.3 (TC)
(MAINTAINABILITY)	= 0.3* (CS) + 0.4* (VS) + 0.1* (MO) + 0.1* (SD) + 0.1* (SI)
(VERIFIABILITY)	= 0.4* (VS) + 0.2* (MO) + 0.2* (SD) + 0.2* (SI)
(EXPANDABILITY)	= 0.3* (AT) + 0.1* (GE) + 0 (VR) + 0.2 (MO) + 0.2 (SD) + 0.2 (SI)
(FLEXIBILITY)	= 0.1* (GE) + 0.3* (MO) + 0.3* (SD) + 0.3* (SI)
(INTEROPERABILITY)	= 0.2 (CL) + 0.2 (FO) + 0.2 (ID) + 0.2 (SY) + 0.2 (MO)
(PORTABILITY) N/A	= (ID) + (MO) + (SD)
(REUSABILITY) N/A	= (AP) + (DO) + (FS) + (GE) + (ID) + (ST) + (MO) + (SD) + (SI)

NOTE: () = Score
 * = Changed

was raised to 0.3 as this is the only one of the five criteria with no conflicts. For flexibility, weighting values for the other three criteria were raised to 0.3. Each is only in conflict with efficiency; and the factor goal is only good.

Weighting formulas for maintainability and verifiability were modified to compensate for conflicts of modularity, self-descriptiveness, and simplicity with the factor efficiency. Weighting values were lowered for these three criteria and were raised for consistency and visibility, which have no conflicts.

The revised set of weighting formulas communicate the needs of the system and user to the developer and incorporate special considerations based on the combination of quality attributes selected.

4.3 SELECT AND QUALIFY QUALITY METRICS

Selecting and qualifying quality metrics is the third of three procedures for identifying software quality requirements (see Fig. 4.0-3). In this procedure, specific metrics and metric elements are selected to be used for evaluating achieved quality levels. Any peculiarities of the system and application which affect metrics require qualification of those metrics. This procedure should be performed by personnel familiar with the application, the system, the software, software development methodology, and QM technology.

This procedure consists of two steps:

- a. Identify metrics.
- b. Select and qualify metric elements.

4.3.1 Identify Metrics (Step 1)

Step 1 is to identify all metrics which are attributes of each criterion which was assigned a weighting value greater than zero in the previous procedure (see Sec. 4.2). The applicable criteria for the surveillance and identification example are listed in Table 4.2-2. All attribute metrics can be identified using Table 3.3-1, which lists all metrics for each criteria.

4.3.2 Select and Qualify Metric Elements (Step 2)

Step 2 is to select specific metric elements which will be used for scoring system and software products. Metric elements are listed in Appendix A on metric worksheets, in question format. The questions are generally applicable to command and control applications. Some questions may not be applicable to a specific command and control application. When selecting metric elements, include system-unique and design-unique considerations such as:

- a. Whether system processors are configured as a centralized processor distributed system, or network.
- b. Whether there is any parallel or concurrent processing.
- c. Whether software is application or executive software.

All metrics and metric elements that are attributes of the weighted criteria in Table 4.2-2 were selected to be used for scoring the surveillance and identification example, with the following qualifications:

- a. Some of the anomaly management metric elements refer to capabilities that are normally provided by executive software. These metric elements will only be used for scoring software providing those capabilities. Any deletion of metric elements will be determined after release of CSCI specifications to which requirements from the surveillance and identification function have been allocated. This qualification avoids restricting the allocation of technical requirements and allows for changes in computing architecture (e.g., surveillance and identification function implemented in a separate processor).
- b. A separate training function (mission training) has been specified at the system level, and some capabilities referred to in the training metric elements may not be provided by software for the surveillance and identification function. All training metric elements will be used in scoring software for the surveillance and identification function. However, data collection personnel may need to refer to other source material (e.g., documentation for software implementing the mission training function) to answer questions. This approach aids in ensuring that appropriate capabilities are provided, since any deficiencies affect usability of software for surveillance and identification.
- c. Efficiency metrics will be applied selectively to software elements. Scores for some software elements may only be monitored and not used in factor scoring. Rationale

for selecting software elements and applying these metrics will be determined prior to software PDR.

When selecting metric elements do not consider any system or design limitations, such as:

- a. Language capability limitations.
- b. Documentation scheme limitations.
- c. Lack of peripherals.

These limitations are by choice. Scoring should simply reflect the degree of presence of a characteristic.

4.4 ASSESS COMPLIANCE WITH REQUIREMENTS

Assess compliance with requirements in the second of two processes for software quality specification (see Fig. 4.0-1). The acquisition manager should perform this process near the end of each software development phase, just prior to formal review. The purpose of the process is to assess compliance of development products with specified software quality factor requirements in the system specification. Source materials are the requirements specifications and Software Quality Evaluation Report (see App. C) containing factor scoring and analyses for the development phase nearing completion and scoring trends for the development cycle. Results of the manager's assessment are used at the development review; action items may result.

The process consists of four procedures (see Fig. 4.0-4):

- a. Review requirement allocations and evaluation formulas.
- b. Review factor scores.
- c. Review criteria scores.
- d. Review metric and metric element scores.

4.4.1 Review Requirement Allocations and Evaluation Formulas

The purpose of this procedure is to determine the appropriateness of the allocation and assignment of software quality factor requirements and of the derivation and use of evaluation formulas. Paragraph 3.2 of the Software Quality Evaluation Report describes

derivation of software quality evaluation formulas and allocation lists based on the allocation of quality factor requirements to software elements (i.e., CSCIs and units). This paragraph and the requirement specifications should contain sufficient information to enable the acquisition manager to (1) trace the allocation of requirements to applicable software elements and (2) check derived evaluation formulas and allocation lists against requirement allocations.

The following list of items may help determine appropriateness of requirement allocations from a quality factor perspective.

- a. All software quality factor requirements should be allocated to at least one software element.
- b. Each software element to which software quality factor requirements have been allocated should support the parent, system-level function.
- c. Software elements to which software quality factor requirements have not been allocated should not potentially affect the quality aspects specified for the parent, system-level function.
- d. The aggregate allocation of requirements should be complete and consistent with respect to the overall system goals.

Evaluation formulas should simply reflect requirement allocations. All applicable software elements should be included. Use of criteria, metrics, metric elements, and criteria weighting should be consistent with specified requirements.

4.4.2 Review Factor Scores

The purpose of this procedure is to determine whether quality factor scores satisfy factor goal requirements. Paragraph 3.4 of the Software Quality Evaluation Report includes a comparison of quality factor scores with specified goals and an analyses of variations. The acquisition manager should review this paragraph to determine whether identified variations are acceptable.

Scoring variations may be acceptable for several reasons. Metrics may have been applied to a draft release of documentation or code; scores for the draft release may be acceptable, provided that scores for the final release reflect correction of deficiencies. Quality factor requirements may be qualified by cost constraints, and scores may reflect the highest possible score within budget constraints.

Scoring should show an upward trend over the development cycle toward the target goal. Paragraph 3.4 of the Software Quality Evaluation Report includes a scoring trend analyses.

Unacceptable scoring variations should be explored. The acquisition manager should review paragraphs 3.4 and 3.5 of the Software Quality Evaluation Report which include scoring variation analyses and corrective recommendations.

If factor scores are unacceptably low, the cause should be investigated. Possible causes include (1) unrealistic goals, (2) low quality products, and (3) biased scoring. If factor goals are determined to be unrealistically high, the process of specifying quality factor requirements should be repeated and new goals established. If low quality products or biased scoring are suspected, criterion and metric scoring should be reviewed. These reviews are discussed in subsequent sections.

If factor scores are unexpectedly high, the cause should also be investigated. Possible causes include (1) unanticipated synergism among factors, (2) high quality products, and (3) biased scoring. If synergism among factors is suspected, the process for specifying quality factor requirements can be reviewed. High quality products and biased scoring can be confirmed by reviewing criterion and metric scoring.

Factor scores should be reviewed and any scoring patterns identified. Consistently high or low scoring could occur for one factor across software elements or could occur for one software element across factors. Scoring patterns help identify the criteria and software elements that should be reviewed to identify scoring variation causes.

4.4.3 Review Criteria Scores

The purpose of this procedure is to investigate possible causes of factor scoring variations through reviewing criteria scoring. Criteria-level scoring involves both the score for the criterion and the weighting value assigned in the factor weighting formula. Weighting values used for scoring should be consistent with specified requirements. Weighting values for each factor formulas should sum to 1.0.

Criteria scores should be reviewed to identify scoring patterns. If a criterion score is low for some CSCIs and not others, the cause could be conflict with a factor. This situation should have been anticipated during the specification process. Other scoring patterns should be investigated by reviewing scores for attribute metrics.

4.4.4 Review Metric Scores

The purpose of this procedure is to investigate possible causes of factor scoring variations through reviewing metric and metric element scoring. Possible causes of high and low scores are *product quality and biased scoring*.

Scores can be biased by measurements of software characteristics that never or rarely vary. Metric and metric element scores that are consistently low indicate a problem. If scores are low across all software elements, the cause could be a design or implementation technique widely used by the development contractor. Changing or enforcing development standards can correct this situation. The cause could also be a system or development limitation. For example, the language chosen may lack certain capabilities or the documentation scheme may not provide adequate information. This situation normally occurs by choice. Metric scoring shows which software characteristics are lacking.

If metric and metric elements scores are low for only certain software elements (e.g., units), the likely cause is low quality for those elements. This can be confirmed by reviewing the product. The cause could be practices by the development contractor. Changing or enforcing development standards can correct this situation.

If metric or metric element scores are consistently high (e.g., all scores are 1.0) over all software elements, the measured software characteristic does not vary. This could be the result of good development practices by the contractor or a feature of the chosen language. If the score can never vary (e.g., because of an automatic feature of the compiler), the metric or metric element should be dropped. If the score varies but is significantly higher than other scoring, consider continuing to monitor the metric score but not using that score in factor calculations. Any low scoring would still be visible, but consistently high scoring would not bias results.

If metric or metric element scores are high for only certain software elements (e.g., units), the likely cause is high quality for those elements. This can be confirmed by reviewing the product.

Visibility is perhaps the most significant benefit of reviewing quality scores. A small variation in the range of scoring values (e.g., 0.5 vs 0.7) may not be as significant as the total scoring picture. By using QM technology, the acquisition manager can periodically view the total product from the perspective of any quality attribute that has been measured. And desired changes can be communicated to the development contractor. Scoring results help in identifying deficiencies and enable corrective action early in the development cycle. Individual scores can vary because judgement is exercised when collecting data. The total perspective provided by the aggregate of scoring results helps minimize the significance of any human element.

APPENDIX A

METRIC WORKSHEETS

(The contents of this appendix are in Vol. III, App. A.)

APPENDIX B

FACTOR SCORESHEETS

(The contents of this appendix are in Vol. III, App. B.)

APPENDIX C

SOFTWARE QUALITY EVALUATION REPORT

Appendix C contains the specification of format and content for the Software Quality Evaluation Report document. Information is in data item description (DID) format. The Software Quality Evaluation Report is used to describe results of metric data collection and analysis.

DATA ITEM DESCRIPTION		IDENTIFICATION NO(S)	
		AGENCY	NUMBER
1. TITLE Software Quality Evaluation Report		USAF	
2. DESCRIPTION/PURPOSE The software quality evaluation report contains a quantitative assessment of achieved software quality factor levels for products released at incremental points during the software development cycle. This report is used by the Air Force to track quality levels and to assess compliance with quality factor requirements in specifications.		4. APPROVAL DATE	
		5. OFFICE OF PRIMARY RESPONSIBILITY	
		6. DOC REQUIRED	
		7. APPROVAL LIMITATION	
3. APPLICATION/INTERRELATIONSHIP The software quality evaluation report describes the results of metric data collection and analyses. A report is normally prepared near the end of each software development phase. Each report should contain metric data and data analyses to address each software quality factor requirement specified in the system requirements specification.		8. REFERENCES (Mandatory to cited in Year 10)	
		MCL NUMBER	
10. PREPARATION INSTRUCTIONS			
<p>1. <u>General Requirements.</u> The software quality evaluation report shall describe results of metric data collection and analyses. Data analyses information shall include correlation of metric scores to factor scores for each software quality factor requirement. Raw metric scores and factor scoring trends shall be included.</p> <p>2. <u>Detailed Requirements.</u> For convenience in describing the minimum essential content, the following paragraphs show a normal format for presentation of material. In the following description, paragraph headings and numbers indicate the general nature of the topic and are minimum mandatory requirements.</p> <p>a. <u>Section 1.0 - Introduction.</u> This section shall describe the purpose and scope of the report.</p> <p>b. <u>Section 2.0 - References.</u> This section shall list both government and non-government references and shall include identification of system/software products used as source material for metric data collection.</p> <p>c. <u>Section 3.0 - Software Quality Evaluation Data.</u> This section shall describe detailed results of metric data collection and analyses and shall identify variations from software quality requirements.</p> <p>(1) <u>Paragraph 3.1 - Product Source Material.</u> This paragraph shall describe the software development phase and system/software products used as source material for collecting metric data.</p>			

DD FORM 1664

S/N-0102-018-4000 PLATE NO. 18448

PAGE 1 OF 3 PAGES

U.S. GOVERNMENT PRINTING OFFICE: 1971-716,877/1000 2-1

D-5517

Software Quality Evaluation Report

(2) Paragraph 3.2 - Requirement Allocation Relationships. This paragraph shall identify and describe the derivation of relationships used for scoring based on the allocation of quality factor requirements to software elements (CSCIs and units). Formulas and lists should be used. For example, $Qsfl = (Qf1 + Qf2 + \dots + Qfn)/N$, where:

Qsfl is the quality factor score for system-level function 1,
Qf1 is the quality factor score for software element 1,
Qf2 is the quality factor score for software element 2,
...
and Qfn is the quality factor score for software element n.

One formula is required for each software quality factor of each system-level function for which software quality factor requirements have been specified. This paragraph shall also identify the specific relationships (criteria and metrics to factors) which were used to calculate software quality factor scores.

(3) Paragraph 3.3 - Data Collection. This section shall describe results of metric data collection and reduction and shall include descriptions of:

- (a) Selection and use of metric worksheets to collect metric element data.
- (b) Selection and use of metric scoresheets to compute metric scores, criterion scores and factor scores.

(4) Paragraph 3.4 - Data Analyses. This section shall describe results of metrics data analyses and shall include descriptions of:

- (a) Computation of quality factor scores for each system-level function.
- (b) Comparison of metric scoring with specified quality factor requirements (goals) and analyses of variations from requirements. *Causes and remedies shall be explored for each variation.*
- (c) Trend analyses, showing software quality factor scoring trends with respect to software development phases.

(5) Paragraph 3.5 - Recommendations. This paragraph shall provide the following:

- (a) Explanations and rationale for scoring variations.
- (b) Recommendations for correcting scoring variations.

Software Quality Evaluation Report

d. Appendix A - Summary Information. This section shall be included as an appendix to the software quality evaluation report. It shall include textual and pictorial material to elaborate and refine material presented in section 3, Software Quality Evaluation Data. These items shall include tabular representations of:

- (1) Software quality factor requirements allocation to software elements.
- (2) A comparison of software quality factor scoring with specified requirements.
- (3) Quality Criteria scoring for each factor.
- (4) Quality Metric scoring for each criteria.

e. Appendix B - Factor Scoresheets. This section shall be included as an appendix to the software quality evaluation report. It shall contain the scoresheets with scores for all applicable factors, criteria, metrics, and metric elements.

f. Appendix C - Metric Worksheets. This section shall be included as an appendix to the software quality evaluation report. It shall contain the metric worksheets with answers to all applicable metric element questions.



MISSION of Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C³I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

END

FILMED

6-85

DTIC